Algo

Date

1. Notes
2. Corman — Introduction to Algo.
3. Google

⇒ Syllabus Algo :⇒

1. Analysing Algo (T.C. && S.C)
2. Divicle And conquer
3. Greedy
4. Dynamic programming
5. Hashing, sorting
6. P- NP, NPH, NPC.
   - NPH: hard
   - NPC: complete

(1.) Analysing Algo :—

Defination of algorithm :-

It is a combination & sequance of finite steps to solve a particular problem.

```
ATN ( )
{
1. Take 2- no's (a,b)
2. c = add (a,b)
3. return (c);
}
```

→ without any line if ans is unaffected, remove that
→ sequance → can not interchange
→ finit steps should be there.
← solution should be these.

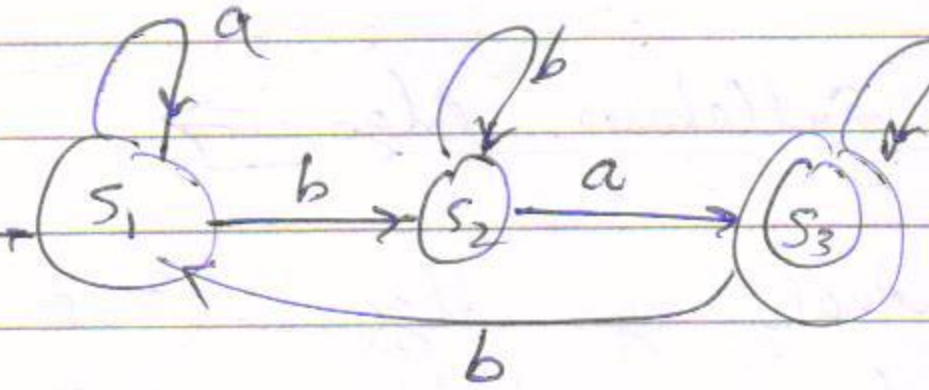
small doubt → 60

ambigouse → confusion
not should be there

Date

⇒ Properties of Algorithm :-

1. It should produce at least one output. = fundamental
2. It should take zero or more input.
3. It should be deterministic.
4. It is independent of programming language.
5. It should terminate after finite time.



⇒ Steps to Construct Algorithm :-

1. Problem defination
2. ✓ Design the algo. {means designing algo, choose one b/w these 3} → Div & Conq. [Divide & Conquer], D.P. [Dynamic Prog.], Greedy
3. Draw flow chart.
4. Verification.
5. (Using any language) Implementation (or) Coding.
6. ✓ Analysing algo.

⇒ Analysing Algorithms-

CPU time
Spee -

⇓

1. Time ⇒ CPU - time —→ have more priority
2. Space ⇒ RAM - space

(1) Time complexity:

$T(P)$ = Compile time (P) + Runtime (P)

$T(P) = C(P) + R(P)$

$C(P)$ → compiler → Software → Programming Language of compiler

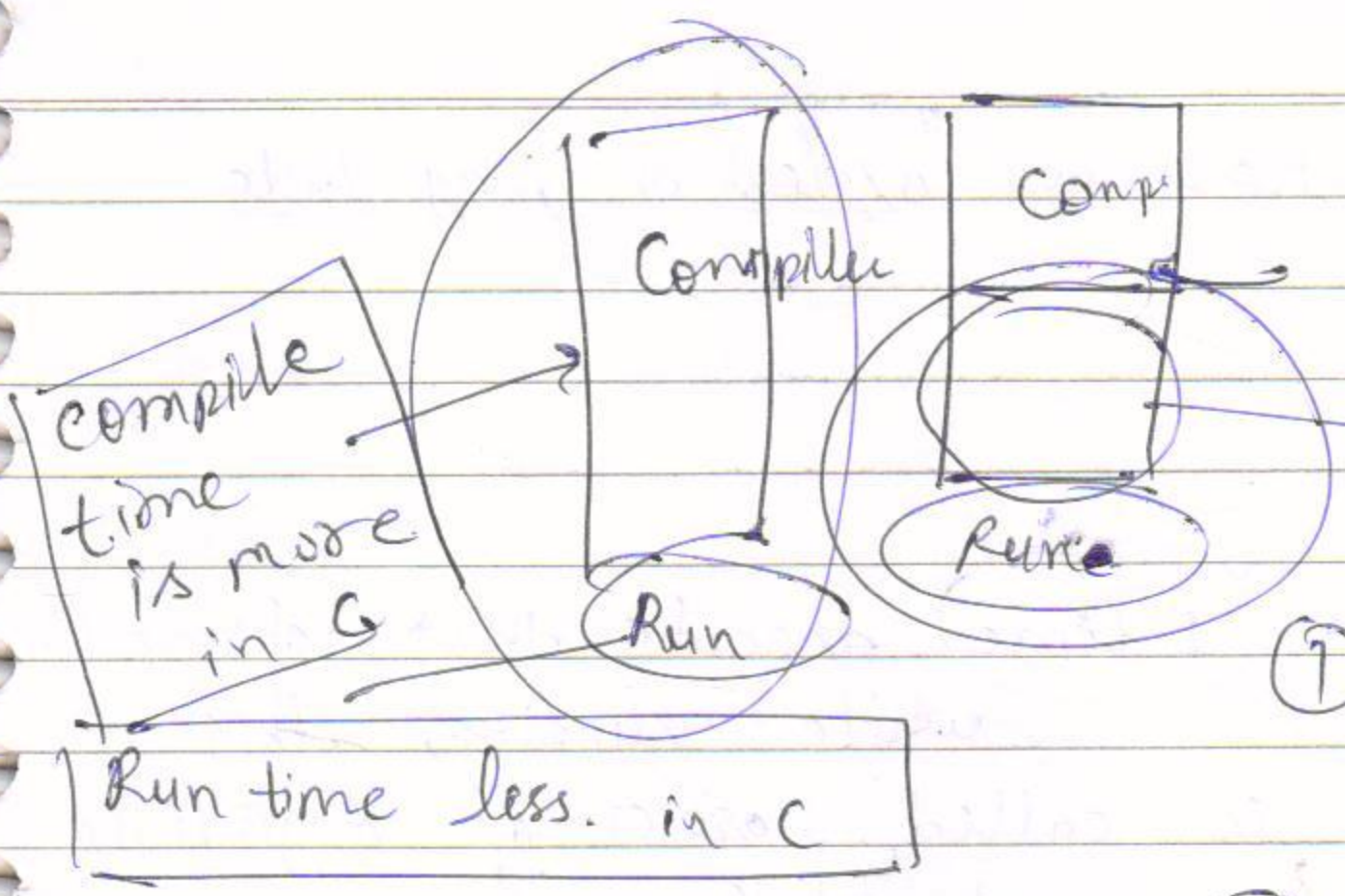
$R(P)$ → CPU → Processor Type → in which compiler written & compilation process completed

C, lang → ↑ faster Programming languages

java → ↓ slow programming language

C Java

Date



intermediate code (Byte code)

Run - jave

① { (machine independent) or (platefarm independent.) }

② compile time of jave less

③ Run time of java More

Types of Analysis

Totaitly dependent on programmer.

Relative Analysis

① Programming Language / (P/L) of compiler. & Type of processor dependent analysis.

② System to system analysis is changing

③ Always ~~correct~~ exact answer.

Absolute Analysis

① P/L of compiler and type of processor Independent Analysis

② System to system analysis is not changing.

③ It will give approximate answer.

Pros

time complexity = $O(n^2)$

$C \cdot n^2$ — depend on programmer logic

Date

⇒ ⇒ Absolute Analysis:-

⇒ It is a determination of "order of magnitude" of statement.

①
```
main()
{
    x = y+z;
}
```
⇒ 1 time execute this statement while running, this is called order of magnitude of statement.

→ ~~O(n)~~ O(1)

②
```
main()
{
(1.) x = y+z;                          ——— (1) time
    for(i=1; i<=n; i++)
    {
(2.)  x = y+z;                         ——— n times
    }
}
```
time complexity = n+1 = O(n)

③
```
main
{
1.  x = y+z;                           ——— 1
    for(i=1; i<=n; i++)
2.      x = y+z;                       ——— n
    for(i=1; i<n; i++)
    {
        for(j=1; j<=n; j++)
        {
3.          x = y+z;                   ——— n x n = n^2
        }
    }
```
↳ find highest power term.

⇒ Locality of referance →

Date

The pages or things that frequenly refereced.

① temporal Locality of referanance. (previous thing..)

② Spacial lo " " "

next one

1
2
3
4
5
6
7
8

2 → temporal

4 → 95% → 5 → special.

4 → 5% → jump. → 100

Note: in cache memory temporal & special locality of reference will be kept

⇒ Time Complexity :—

finding largest loop.

④

```
main ()
{
    for (i=1 ; i ≤ n ; i++)                    n
    {
        for (j=1 ; j ≤ n/2 ; j++)              i=1
        {                                      ...n/2
            x += y+z ;   ⟶   n · n/2 = 1/2 n²
        }                                      leave constant
    }
}
```

$= O(n^2)$

(5) main ( )

```
{
    for (i=1 ; i<=n ; i++)
    {
    for(j=1 ; j<= i ; j++)
        {
        for ( k=1; k<= 133; k++)
        {
          x = y+z;
        }
    }
}
```

i=1	i=2	i=3 n	n
—k=1	j=1 2	3 n	n
k=133	1...133 1...133		k=1; 133

$$T = 1(133) + 2(133) + 3(133) + \cdots + n(133)$$

$$= 133[1 + 2 + 3 + \cdots + n]$$

$$= 133\left[\frac{n(n+1)}{2}\right] = 133\left[\frac{n^2+n}{2}\right]$$

$$= \frac{133}{2}(n^2 + n)$$

Time complexity $= O(n^2)$

(6) main ( ) ;

Date

$n \times n^2 \times n^3 = n^6$ ×

```
{
  for (i=1; i<=n; i++)
  {
     for(i=1; i<=n^2; i++)
     {
        for (i=1; i<=n^3; i++)
        {
  1.       x = y+z;
        }
     }
  }
```

n^3

$i = n^3$

$i = 1$	$i = 2$
$j = 1 \ldots n$	$j = 1 \ldots n^2$
$i = 1 \ldots n^3$	

because $i = n^3$ bcoz when once $i = n^3$ it will false for loop ① & ② and finally statement 1 will be executed n^3 times.

$TC = O(n^3)$

(3)

Date

(7) main()
{
while (n>1)
{
$n = n/2$
}
}

$n \geq 2$ —— 3

$n = 1$

1	2	3	4
0	1	① $3/2$ ② $\frac{3}{2 \times 2}$ $\frac{3}{4}$	① $\frac{4}{2} =$ ② $\frac{2}{2} = 1$

32
① ↓
$32/2 = 16$
↓
② T. 16 > 1
$\frac{16}{2} = 8$
↓
③ T 4 > 1
$\frac{8}{2} = 4$

4

$8 = 3$ (2^3)
$16 = 4$ (2^4)
$32 = 5$ (2^5)
$64 = 6$ (2^6)
$128 = 7$ (2^7)

$TC = O(\log_2 n)$

(8) main()
{
while (n>1)
{
$n = n/3$; —— $O(\log_3 n)$
}
}

multiply (or) { divide → log
addition (or) { subtract → divide }

Date

```
while (n>1)
{
  n = n-1;
}
```

$\frac{n}{2} \Rightarrow O(n)$

$\Rightarrow$

```
main ()
{
  while (n>1)
  {
    n = n/2;
    n = n/3;
  }
}
```

$n = \frac{n}{6}$

$\log_6 n$

$n = 100$

$100 > 1$

$= 50 =$

$\frac{50}{3} = 16$

$\frac{100}{6}$

$= 16$

$\Rightarrow$

```
main ()
{
  while (n>1)
  {
    n = n/2;
    n = n-2;
  }
}
```

$\log_2 n$

$n = 100$

$100 > 1$

$50 > 1$

$21 > 1$

$\frac{100}{2} \geq 50$

$50 - 2 = 48$

$\frac{48}{2} = 24$

$n-2 = 22$

$\frac{22}{2} = 11$

multipling by 2 =

Date

Q. main
{ i=1
while (i<n)
{
i = 2*i ;
}
}

n=16

1 < 16 ①

2 < 16 ②

4 < 16 ③

8 < 16 ④

16 <

increasing by 2.

so $\log_2 n$

$n \geq 2$

2 1000

```
main ()
{ while (i<n)
{
i = i+100;
i = i+200
}
}
```

$\left(\frac{n}{2100}\right)$

O(n)

```
while (i<n)
{
i = 2*i ;
i = i+100 ;
i = 3*i ;
i = i+2000 ;
```

i = i*6 ⟹ $\log_6 n$

Date

```
main ()
{
    for (i=1 ; i <= n ; i++)
    {
        for (j=1 ; j<=n ; j = (2*j))
        {
            for (k=n ; k>=1 ; k = k/2)
            {
                x = z+y ;
            }
        }
    }
}
```

$n * \log_2 n * \log_2 n$

$= n (\log_2 n)(\log_2 n)$

$T.C. = n (\log_2 n)^2$

i = 1

j = 1 -- 2.

k = 100 ... 50 ... 25 1

$\log_2 n$

Case

if ⇒

for (i=1 ; i<=n ; i = 2*i) ⟶ $\log_2 n$

18/10

Q. Find the time complexity of following programm. Date

:⇒
```
A ()
{
    n = 2^2^k        where k>0
    for (i=1; i<=n; i++)
    {
        j=2;
        while (j<=n)
        {
            j = j^2
        }
    }
}
```

j^2 is increamenting very fast so while loop will execute

$n = (2^2) = 4$

$k=1$,

$i=1$

$j = 2, 2$

$2 < 4$

① $j = 2 \times 2$

$j = 4$ .2 times

$4 \leq 4$

② 2 times

4 $n(k+1)$

2 2 2 2

8 = 2×4 time

$k=2$

$n=16$

$i=1 \quad n$

$j = 2 \leq 16$

① $4 \leq 16$

② $16 \leq 16$

③ 3 times

$k=2$ $n(k+1)$

16x..

10.

So. for value of k loop will execute $n(k+1)$

$j^{2^k} = n$

apply $\log_2$

$\log(j^{2^k}) = \log n$

$2^k \log_2 j = \log_2 n$

$2^k \log_2 j = \log_2 n$ » $j=2$

$2^k \log_2 2 = \log_2 k$

(a) $O(n^2)$ (b) $O(n \log n)$ (c) $O(n \log \log n)$ (d) $O(n)$

again log ⇒ $2^k = \log_2 n$

$\log_2 2^k = \log(\log_2 n)$

$k \log_2 2 = \log \log_2 2$

in Algo = Constant is of O(1)
↳ may be 0

(13)

Date

$$\left(\left(\left(2^2\right)^2\right)^2\right)^{2\cdot r\cdot k} = 2^{2^k} = n \quad \text{till } n$$

$$\log 2^{2^{2^k}} = \log n$$

$$2^k \log_{2^{2^k}} 2 = \log_{2^{2^k}} n$$

$$2^k = \log n$$ again $\log$

$$\log 2^k = \log(\log n)$$

$$k = (\log \log n)$$

putting value

$$n(k+1) \Rightarrow n(\log(\log n) + k$$

Q if statement was $j = j^2 \Rightarrow j = 2*j$ · then

$$j = 2*j \times 2 \times 2 \times 2 \times 2 \cdots \Rightarrow 2^k \cdot j \quad \text{where } (((j\times 2)\times 2)\times 2)\ldots = k = n$$

$$j 2^k = n$$

$$\Rightarrow 2^k \cdot 2 = n$$

$$\log 2^{k+1} = \log n$$

$$k+1 = \log n$$

$$k = O(\log n)$$

for $k=1$	$k=2$	$k=3$	$k=n$
$n=4$	$n=16$	$n=256$	$n=2^{2^n}$
$i=1\ldots4$	$i=1\ldots16$	$i=1\ldots256$	
/	/	/	
$n(2 \text{ times})$	$n(4 \text{ times})$	$n(8 \text{ times})$	$n(2^k)$
$n(2^k)$	$n(2^k)$	$n(2^k)$	$O(n(\log n))$

Q if statement will be changed $j = j/2$;

⇒ going into infinite loop. ∴

```
while ( j <= logn)
  {
     j= 2*j;
  }
```

k=1	k=2	k=3
n=4	n=16	Date
i=1...n/2	i=1...n	
log2	log2^4	
j<=2 (log)	2<=4 (log n)	
2<=2^1		
4<=2^2 (log n)	4<=4^1 (log n)	n (log log n)
1 time	n (2 times)	3 times
	n (k)	3 (n k)

⇒ Q. `j <= 100`

```
{
  j = j^2
}
```

⇒ log (log_2 100)

= log (constant)

constant

= O(1)

Q. `j <= 100 sqrt(n)`

```
{
  j = 2*j
}
```

Q ⇒ new A(n) = time complexity.

```
{
  if (n <= 2) return;
  else
  return (A(sqrt(n)));
}
```

log(1000) = 33 ; log(10) = 3

A(1000) → A(33) → A(6) → A(2)

√1000 → √33 → √6 → 2

(log log n) fol

T.C}

(a) O(n) (b) O(logn) (c) O(log logn) (d) O(1)

$2 \leq 2$

$n = 1000$

$A(\sqrt{1000}) \quad 2^3 \leq 2^k$

$(1000)^{1/2} = (10)^{3/3}$

$(\frac{1}{2}^0)$ 3^1

$\sqrt{1000} \times 10 \times 10$

5×2

5×20

take always large value

Date

⇒ if $A\sqrt{n}$ changed $A\left(n/2\right)$

log 1000 → 10

log 10

↓

3 ...>

Ans. anther way

$\left(n^{1/2}\right)^{1/2} \cdots i/2$

KP

means → $\log n^{1/2^k} = \log 2$

$\frac{1}{2^k} \log n = 1$

$\log n = 2^k$

$\log \log n = \log 2^k$

$\log \log n = k \cdot 1$

$(\log \log n) = k$

Space complexity :—

2 Bytes everytime

Stack is used in recursion

$(\log \log n)$

```
A (n)
{
if (n <= 2) return;
else
return (n);
}
```

TC = O(1)

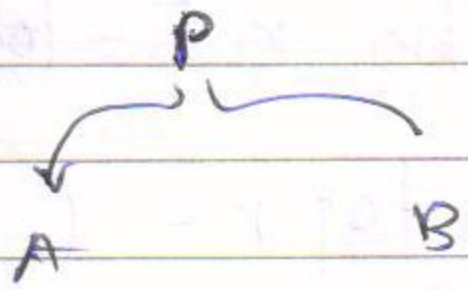
Date

⇒ Asymptotic Notation :-

Goal / Purpose :-

3 Notations are these —

① Big -oh- Notation
② Omega - Notation
(3) Theta - Notation



A = O(B)

↓ best ✓

↘ B take more time

A = Ω (B)

↳ B is taking less time

(B best) ✓

A = Θ B

↳ both → same

Parent child

✓ P = Ω(C)

✓ C = OP

Ques.

$1+2+3---+n = \frac{n(n+1)}{2}$ Date $\frac{n^2}{2}$

$n+n+n+---+n = n^2$

relation b/w $\frac{n^2}{2}$ & n^2

$\frac{n(n+1)}{2} = O(n^2)$

S = Salary Expences = E

$E = O(S)$

* Valid for positive integers :—

Let $f(n)$ && $g(n)$ be two positive functions

$(n-100)$ is positive function bcoz for larger value.

$(n-10000)$ → also positive function bcoz constant

eg of negative function | $10000 - n$ |

it will be negative fun.

integer

Z

Z^- $(-1, \dots, -\infty)$ — 0 — Z^+ $(1, \dots, n \to \infty)$

Date

Big-oh Notation :-

$$f(n) = O(g(n))$$

equal

$f(n)$ is order of $g(n)$

iff

$$f(n) \leq c \cdot g(n) \quad \forall n, n > 0$$

& $(n \geq n_0)$ where n is +ve integers and c is a +ve constant ~~which is greater~~ $(c > 0)$

1, 1.01, 1.002 ...

1, 2, 3 ...

$2n = O(n)$

$2n \leq c \cdot n$

↓ constant 2 or 3 or

(eg) $f(n) = n + 10$

$g(n) = n$

$\Rightarrow$ $f(n) = O(g(n))$

iff

$n + 10 \leq c \cdot n$, $\forall n, n \geq n_0$

$n + 10$ $\quad$ $c \cdot n$

$\Downarrow$ $\quad$ $\Downarrow$

2 $\quad$ 2

$n + 10 \leq c \cdot n$ $\quad$ $\forall n, n > n_0$

$\Downarrow$ $\quad$ $\Downarrow$

2 $\quad$ 10

$(n_0 = 10)$ for this $\quad$ $n + 10 = O(n)$

↳ is greater

Date

eg Q.

$f(n) = n$

$g(n) = n+10$

$\Rightarrow n = O(n+10)$

$\Downarrow$

$n \leq \underbrace{c.(n+10)}_{\downarrow}$, $\forall n$, $n \geq n_0$

$\downarrow$ $\downarrow$ $\downarrow$

1 already greater 1

eg 3 $f(n) = n^2$

$g(n) = n$

$\therefore$ $n^2 = O(n)$

$n^2 \leq c.n$ $\forall n$, $n \geq n_0$

$\Downarrow$

$n \rightarrow$ n is not possible

$n^2 \neq O(n)$

eg $n^2+n+100 = O(n^2)$

$n^2+n+100 \leq c.n^2$

$\downarrow$ $\Downarrow n_0$

$100+10+1 = 3 \times 100 \leftarrow 10$

$= 300$

$2 < 0$

(2) Omega - Notation :- Ω

Date

$f(n) = \Omega(g(n))$

or

$f(n)$ is omega of $g(n)$

iff

~~$f(n) \geq C \cdot g(n)$, $\forall n$,~~

$f(n) \geq C \cdot g(n)$, $\forall n$, where n is +ve integer

&& $n \geq n_0$

where C is a +ve constant.

Ex. ① $f(n) = n$

$g(n) = n + 10$

$n = \Omega(n+10)$

$n \geq C \cdot (n+10)$, $\forall n$, $n = n_0$

for $\frac{1}{2}$

$n_0 = 10$

$n = \Omega(n+10)$

$C = \frac{1}{2}$

$n_0 = 10$

Date

eg 2:- $f(n) = n+10$

$g(n) = n$

$n+10 = -2(n)$

$n+10 \geq c \cdot n \quad \forall n, \forall n \geq 1$

$\Downarrow$

1

$n_0 = 1$

eg 3 $f(n) = n$

$g(n) = n^2$

$n = -2(n^2)$

$n \geq c \cdot n^2$

$\Downarrow$

n — this is function, we can't use function

n — but this is not constant

not possible.

system fails.

$n \neq -2(n^2)$

(3) Theta - Notation :-

Date

$f(n) = \theta(g(n))$

(or)

$f(n)$ is theta of $g(n)$

iff

1. $f(n) \leq c_1 \cdot g(n)$ && 2. $f(n) \geq c_2 \cdot g(n)$ } where c_1 & c_2 are +ve constant & $n \geq n_0$ where n is +ve integer.

eg ①:- $f(n) = n+10$

$g(n) = n$

1. $f(n) = O(g(n))$

$n+10 \leq c_1 n$, $\forall n$ $n = n_0$

$\downarrow$ $\downarrow$

2 10

2. $f(n) = \Omega(g(n))$

$(n+10) = 2(n)$

$n+10 \leq 2n$ $n+10 \geq c_2 n$ $\forall n$ $n = n_0$

$\downarrow$ $\downarrow$ $\downarrow$

1 1 1

$\downarrow$ $\downarrow$

1. $f(n) = O(g(n)) \Rightarrow c_2 = 2$ } $n_0 = 10$.
2. $f(n) = \Omega(g(n)) \Rightarrow c_2 = 1$ }

Ex 2:- $f(n) = n$

$g(n) = n$

Date

1. $f(n) = O(n)$

$n \geq c_1 n$, $\forall n_0$, $n = n_0$

$\Downarrow$ $\Downarrow$

1 1

2. $f(n) = \Omega g(n)$ → (1)

$n \geq c_2 \cdot n$ n_0

$\Downarrow$ $\Downarrow$

1 1

$n = \Theta(n)$ $c_1 = 1$ $n_0 = 1$

$c_2 = 1$

Ex 3 $f(n) = n$

$g(n) = n^2$

1. $n = O n^2$

$n \leq c_1 n^2$ $\forall n$ $n_0 = 1$

$\Downarrow$

1

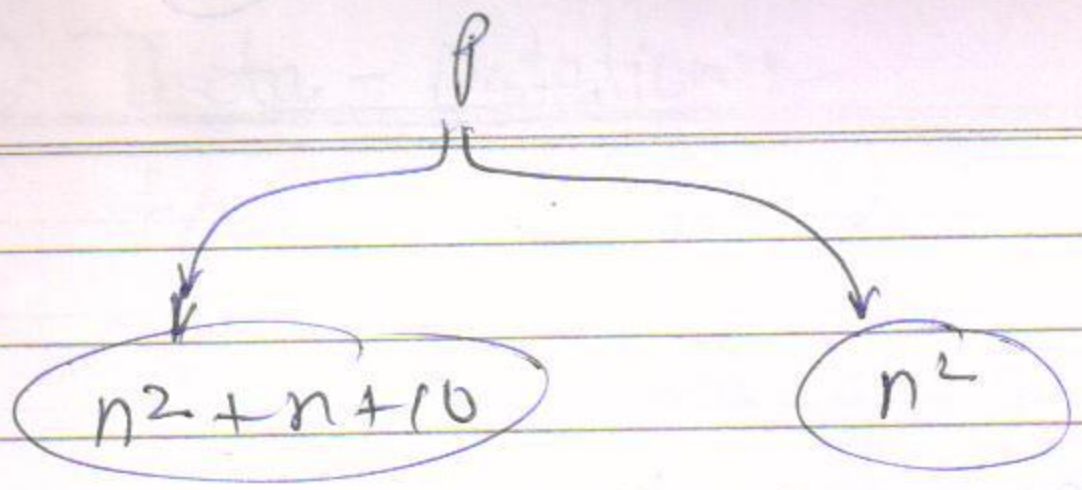
2. $n \geq c_2 n^2$

$\Downarrow$

$1/n$ ✗

so $n \neq \Theta(n^2)$.

Date



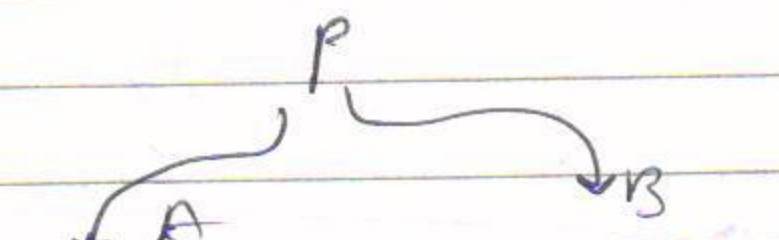
(1) $n^2+n+10 = O(n^2)$

$n^2+n+10 \leq c_1 n^2$

(2) $n^2+n+10 = \Omega(n^2)$

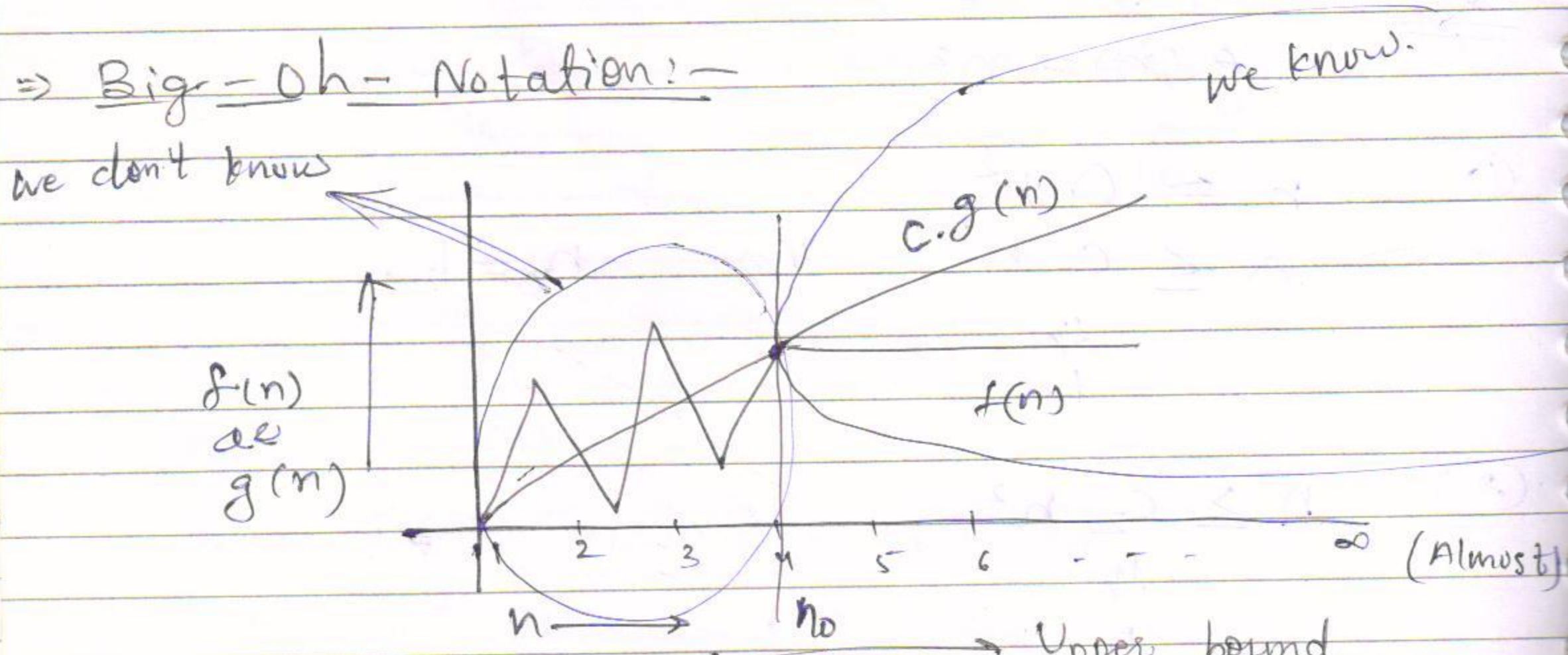
$n^2+n+10 \geq c_2 n^2$

$\Rightarrow$ $n^2+n+10 = \Theta(n^2)$



Relation :- $A = O(n^3)$

$\Rightarrow$ Big-Oh- Notation :-

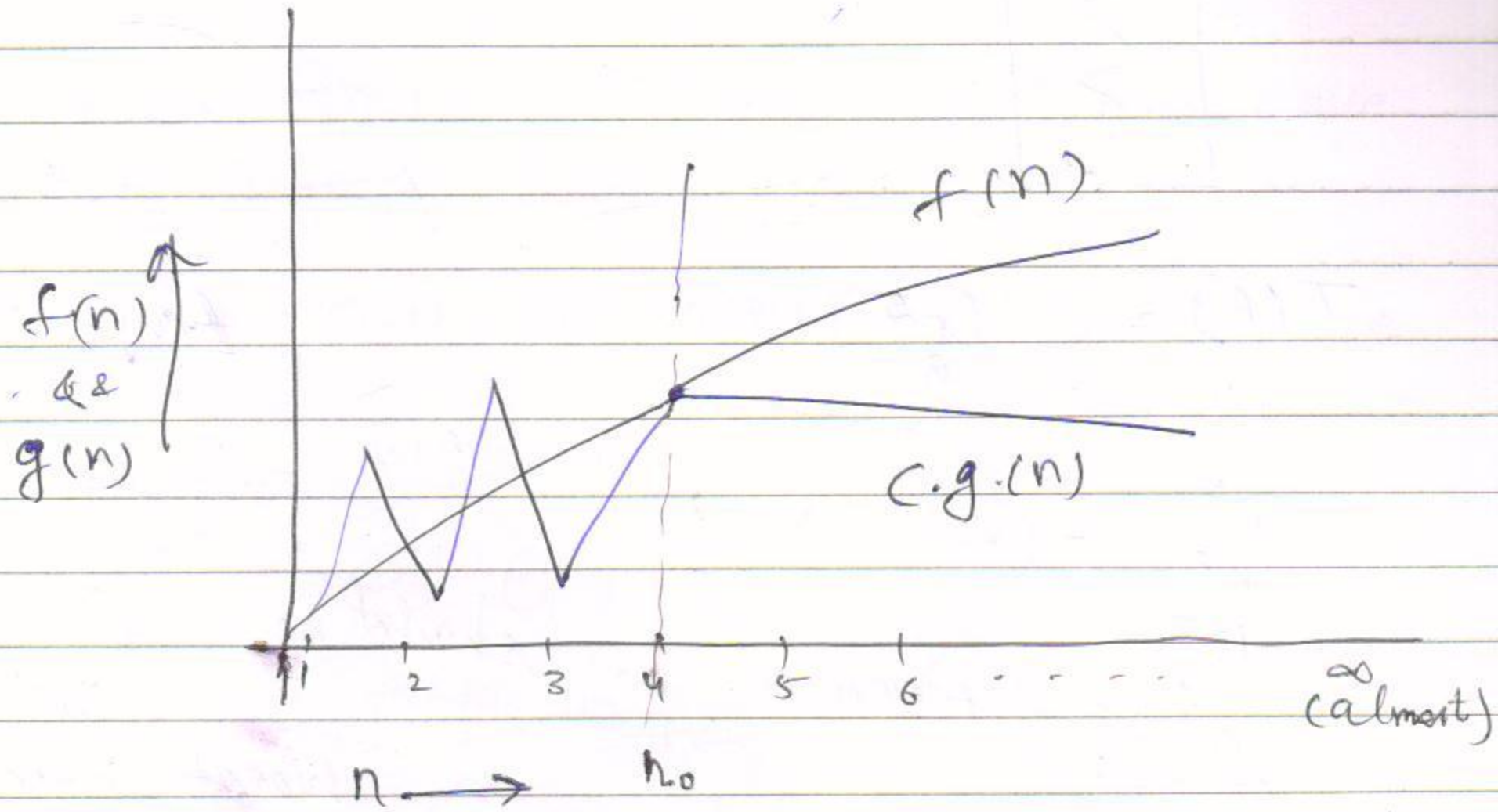


Big Oh " mean $f(n) = O(g(n))$

$n \leq c \cdot g(n), \forall n, \quad n = n_0$

Date

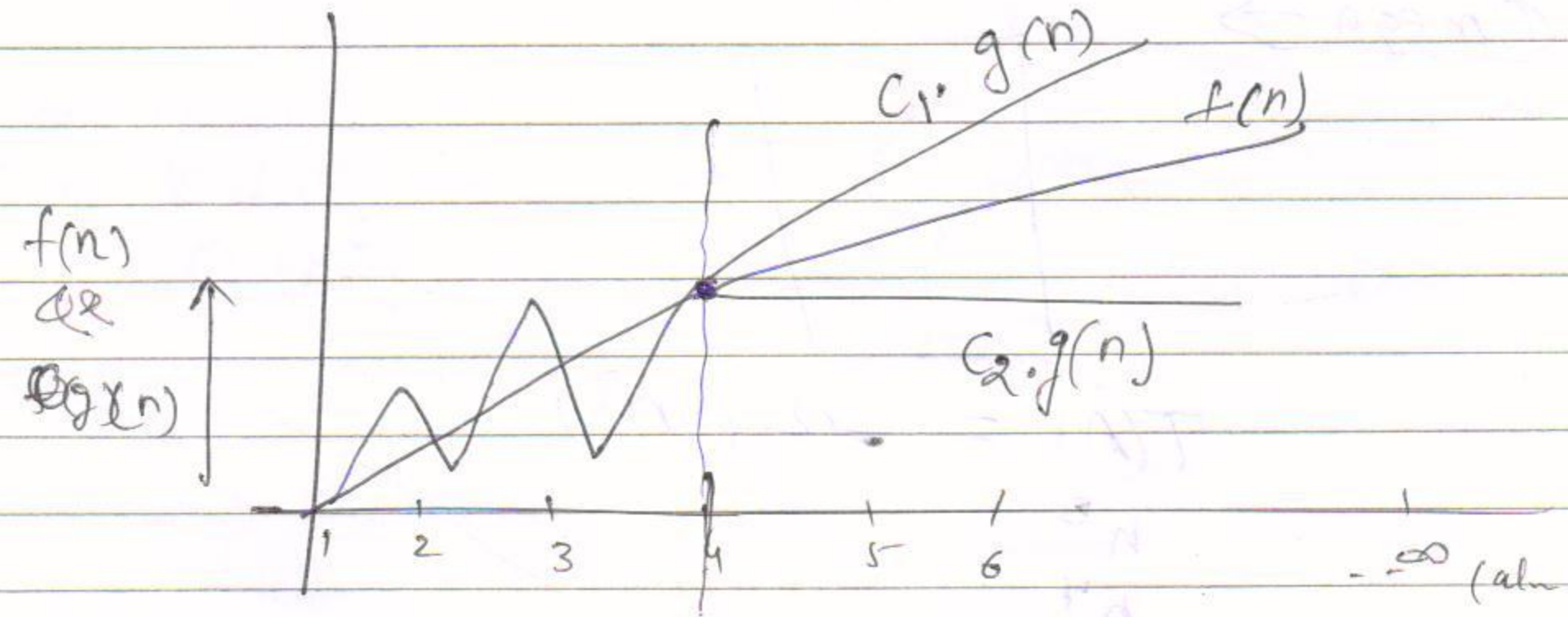
Omega Notation :-



$f(n) = \Omega(g(n))$ ⟶ Lower bound

$f(n) \geq C.g(n) \quad \forall n, \; n = n_0$

⇒ Theta Notation :-



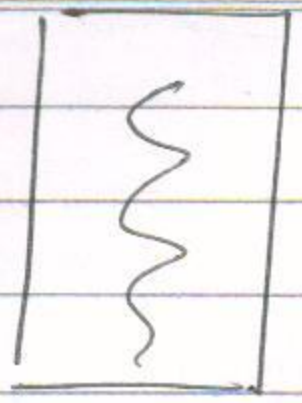
$f(n) = \Theta(g(n))$ ⟶ both upper bound && Lower Bound

1. $f(n) = \Theta\, g(n)$
   $f(n) \leq C_1 . g(n) \quad \forall n, \; n = n_0$

(2) $f(n) = \Omega\, g(n)$

Date

A



$T(A) = O(n^5)$ $\quad f(n) = O(g(n))$

n^4

n^3

n^2

1

n^6 X

upper bound

maximum not greater than / should be

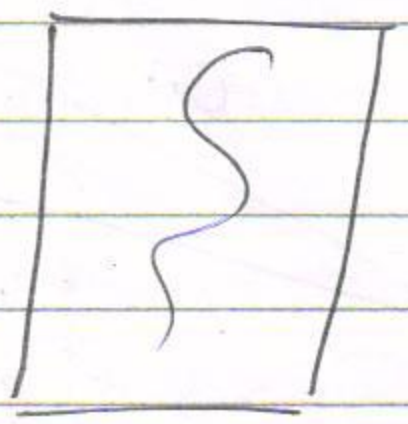
that is worst case

Note

So To find out worst case Big-oh-notation is used.

Omega ⇒

A



$T(A) = \Omega(n^3)$

n^3

n^4

n^5

n^6

$T(4\text{-B.Tech}) = \Omega(4)$

4

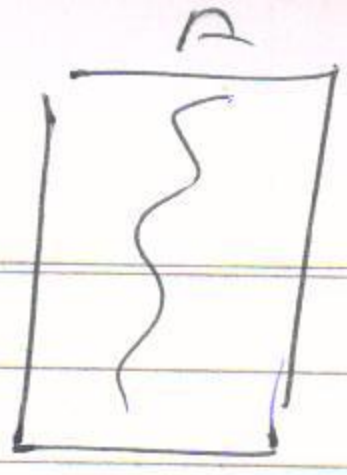
5

6

7

Date

Theta - Notation :-



$T(A) = \Theta(g(n))$

$T(A) = 2(g(n))$ ✗ why write 2.

$T(A) = \Theta(g(n))$ ← write once.

Gate

(a) ~~$\Theta(g(n))$~~ $O(n^2)$

(b) ~~Θ $2(g$~~ $2(n^2)$

✓(c) $O(n^2)$ Ans

(d) None of these

⇒ Conclusion :-

Big-oh - Notation :- ($\leq$)

$n^2 = O(n^2)$ ⇒ Tightest upper bound ← worst case.

$n^2 = O(n^3)$ ⇒ Not Tightest upper bound.

$n^2 = O(n^{10})$ - NO TUB.

$A = O(B)$

⇓

~~TUB~~ Upper bound

it may be

Tightest ⇒ equal to

Not Tightest ⇒ greater

Date

⇒ Small - oh (<)

$n^2 = o(n^2)$ ✗ (=) is not allowed only < is allowed.

$n^2 = o(n^3)$ ✓ NTUB

$= o(n^4)$ Not TUB

$= o(n^{10})$ Not TUB

$A = o(B)$

⇓

not tightest upper bound.

⇒ Omega - Notation :- ($\geq$)

$n^3 = \Omega(n^3)$ ⇒ Tightest lower bound

$= \Omega(n^2)$ ⇒ NoT TLB

$\rightarrow \Omega(n)$ ⇒ NOT TLB

$A = \Omega(B)$

⇓

LB

may be ⇒ TLB Not TLB

⇒ Small - omega (ω) (>) :-

$n^3 = \omega(n^3)$ ✗

$n^3 = \omega(n^2)$ ✓ NTLB

$= \omega(n)$ ✓ NTLB

~~A = B~~ $A = \omega(B)$

⇒ Theta - Notation :-

Date

$n^3 = O(n^3)$ — TUB

&&

$n^3 = \Omega(n^3)$ — TLB

↓

$n^3 = \Theta(n^3)$

↓

$A = \Theta(B)$

↓

TLB && TUB

Q.

$n(n-1).(n-2)(n-3)(n-4)\cdots\cdots 1 = n!$

$n \cdot n \cdot n \cdot n \cdot n$ $\quad n = n^n$

$n! = O(n^n)$

$4! = O(4^4)$

$24 = O(256)$

$5! = O(5^5)$

$5! = o(5^5)$

$n+n+n+n = O(n)$

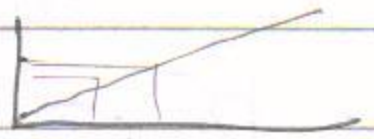
⇒ Types of Time Complexity :-

Date

(1.) Constant Time complexity ⇒ $O(1)$

(2.) Logarithmic Time Complexity ⇒ $O(\log n)$

(3.) Liner Time complexity ⇒ $O(n)$



(4) Quadratic Time complexity. ⇒ $O(n^2)$

(5.) Cubic T. C ⇒ $O(n^3)$

(6.) Polynomial T.C. ⇒ $O(n^k)$ $k>0$

where k. is a constant

TC = $O(n^k)$ for loops will be the

1000 for loops

A

n^k, $k>0$

P class.

| h

Const

c^n $c>1$

2^n $c>1$

3^n

(7). Exponential T.C. ⇒ $O(c^n)$

where c is a constant & $c>1$

Expunential

Date

① $n = 2^{100000}$

$\log n = 1,00,000 = 2^{17}$

$\log(\log n) = \log 17 \Rightarrow 17$

$\sqrt{\log n} = 300$ = greater the log

or $\log n (\log n)$ $\sqrt{\log n}$

$\log(\log(\log n))$: $\log(\log n)^{1/2}$

$\log(\log(\log n))$: $\frac{1}{2}\log n$

Q:- $(n+k)^m = O(n^m)$ where k & m are constant.

Ans (a) $(n+k)^2 = n^2 + k^2 + 2nk$

$\leq n^2 + n^2 + n^2$ max

$\leq 3n^2$

$\leq c_1 n^2$

$(n+k)^3 = n^3 + k^3 + 3n^2k + 3nk^2$

$= n^3 + n^3 + n^3 + n^3$ max

$= 4n^3$ $= 4n^3$ $= O(n^3$

$(n+k)^m \leq (m+1) \cdot n^m$

$= O(n^m)$

Date

① const T.C ⇒ O (1)

O [log (log n)]

O (√log n)

② Log TC ⇒ O (log n)

O (n (log n))

(n √n)

③ liner ⇒ O(n)

O (n log n)

n √n

④ Quadratic T.C ⇒ O(n²)

Date

8.) $2^n < n^n \Rightarrow 2^n = o(n^n)$ — strictly

$\Downarrow$ $\nexists x$ — not always possible

$2^n = O(n^n)$

$n! < n^n \Rightarrow n! = o(n^n)$

$n! > 2^n \Rightarrow n! = \omega(2^n)$

$\rightarrow 2^n \quad n^n \quad \log$

$n < n \log n$

now $\Rightarrow n < (\log n)^{\log n}$

apply log for $n > (\log n)^{\log n}$

$\log n > \log(\log n)$

$\log \log n = \log [(\log n)^{\log n}]$

$\log n =$

$n! = o(n^n)$

$2^n = o(n!) = o(n^n)$

Relation

$n > \log n$

$n > (\log n)^2$

$n > (\log n)^3$

$n > (\log n)^{100}$

$n > (\log n)^{100000}$

$n_0 =$ depends

Date

→ $n > \log n$

$n > (\log n)^2$

$n > (\log n)^{100}$

$n < (\log n)^{\log n}$

Gate :-

Q. Check False & True

(i) $100 \cdot n(\log n) = O\left(\frac{n \log n}{100}\right)$

(b) $\sqrt{\log n} = O(\log \log n)$ ⇒ false

$\frac{1}{2} \quad \frac{3}{3}$

(c) if $0 < x < y$ then

$n^x = O(n^y)$

(d) $2^n \neq O(n^k)$ where k is constant & $k > 0$

Ans (i) $100 \cdot n(\log n) \leq n(\log n)$ True

$\log 100 + \log(n \log n) \leq \log[n(\log n)] - \log(1.00)$

(b) $\sqrt{\log n} \leq \log(\log n)$

Date

$\log \cdot (\log n)^{1/2} \leq \log(\log(\log n))$

$\frac{1}{2}(\log n) \leq \log \log \log \cdot$

false

Ans.(a) by Sir.

$100 \cdot n \log n = O\left(\frac{n \log n}{100}\right)$

$100 \cdot n \log n \leq C \cdot n \log n$

$\Downarrow$

$10,000$

Ans.(b.) False

Ans.(C). if, True.

Ans.(D). $2^n \neq O(n^k)$ where k is const

$\downarrow$ $\downarrow$

exponential > constant

always greater. should

$\downarrow$

$\neq 0$

$\rightarrow$ statement is true.

Gate:-

Date

Q Find True and false:

(a) $(n+k)^m = \Theta(n^m)$ where k & m are constant. true

(b) $2^{n+1} = O(2^n)$ true.

(c) $2^{2n} = O(2^n)$ false

(d) $f(n) = O((f(n))^2)$, if $f(n) \to <1$ (false), <0 negative, >1

(a) (i) $(n+2)^{1000} \leq c_1 n^m$ true

$(n+k)^2$

$\leq c_1 n^{1000}$

$= n^2$

(b) $2^{n+1} = O(2^n)$

$2^{n+1} \leq C_1 \cdot 2^n$ True.

(c) $2^{2n} \leq c_1 2^n$ false.

To apply log first remove the common things.

 Date

(d)

By Six :-

(b) $2^{n+1} = O(2^n)$

$2 \cdot 2^n \leq C \cdot 2^n$

$2, 2^n \leq 3 \cdot 2^n$

true

(c) $2^{2n} = O(2^n)$

$2^{n+n} \leq C \cdot 2^n$

$2^n \cdot 2^n \leq C \cdot 2^n$

with any value of C we can not balance

Ans. False

(d) $f(n) \leq c_1 f(n) \cdot f(n)$

$\log 1 \leq c_1 \log (fn)$

let $\Rightarrow$ $f(n) = \frac{1}{n}$, if fraction value is there

$\frac{1}{n} = O\left(\left(\frac{1}{n}\right)^2\right)$ in case $n = (.5)^2$

$2^n \quad 2^{2n}$

$\log \quad \log$

$\log 2^n : \log 2^{2n}$

$n(\log 2) : 2n \log$

$8^{\log_2 4}$ (38)

$a^{\log_b n} = n^{\log_b a}$

Q. find true or false ?

Date

✓ (a) $n^2 \cdot 2^{3\log_2 n} = \theta(n^5)$

✓ (b) $\frac{4^n}{2^n} = \theta(2^n)$

✗ (c) $4^{\log_2 n} = \theta(n^3)$

✗ (d) IMP If $f(n) = O(g(n))$ then $2^{f(n)} = O(2^{g(n)})$ false

(a)(I). (1) $n^2 \cdot 2^{3\log_2 n} \leq c\, n^5$

$\Rightarrow n^2 \cdot n^{3\log_2 2} \leq c\, n^5$

$n^2 \cdot n^3 \leq c n^5$

$n^{3+2} \leq c n^5$

$n^5 \leq c n^5$

II

1

(2) $n^5 \geq c_1 n^5$

II

1.

$(n^5) = \theta(n^5)$ true

Date

Ans (b) $\frac{4^n}{2^n} = O(2^n)$

(1) $\frac{4^n}{2^n} \leq O\, 2^n$

$\frac{(2^2)^n}{2^n} \leq C_1 2^n$

$\frac{2^{2.n}}{2^n} \leq C_1 2^n$

$2^{n+n} \leq$

$\frac{2^n \cdot 2^n}{2^n} \leq C_1 2^n$

(2) $2^n \geq C_1\, 2^n$

$1 \geq 1$

True

Ans (c). $4^{\log_2 n} = O\, n^3$

$2^{\log_2 n^2} =$

$n^{\log_2 2^2}$

(1) $n^2 \leq C_1 n^3$

(2) $n^2 \neq \Omega\, n^3$ False

Ans IMP Qet

Date

(d) If $f(n) = 2n$

$g(n) = n$

$f(n) = O(g(n))$

$2n = O(n) \Rightarrow 2n \leq C_1(n)$

$2^{2n} = O(2^n)$

$2^n \cdot 2^n \leq C \cdot 2^n$

exponential ✗ not possible

Hence

If $f(n) = O\,g(n)$ so false

Q. $f(n) = n^3$ $1 \leq n \leq 10,000$

$= n$ $n \geq 10,000$

$g(n) = n^2$ $1 \leq n \leq 100$

n^3 $n \geq 100$

T ?

(a) $f(n) = O(g(n))$ → Ans. $n_0 = 100$, true

(b) $f(n) = \Omega(g(n))$ $n_0 = 1$ ✗ we can say only about starting from where till ∞ condition should be true.

(c) $f(n) = \Theta(g(n))$ ✗

(d) None of these.

Ans. (a) $n^3 = O(n^2)$

$n^3 \leq C\,n^2$ false

$(10000)^3 = 10,000,000,000$

$n \cdot n \cdot n = n \cdot n$

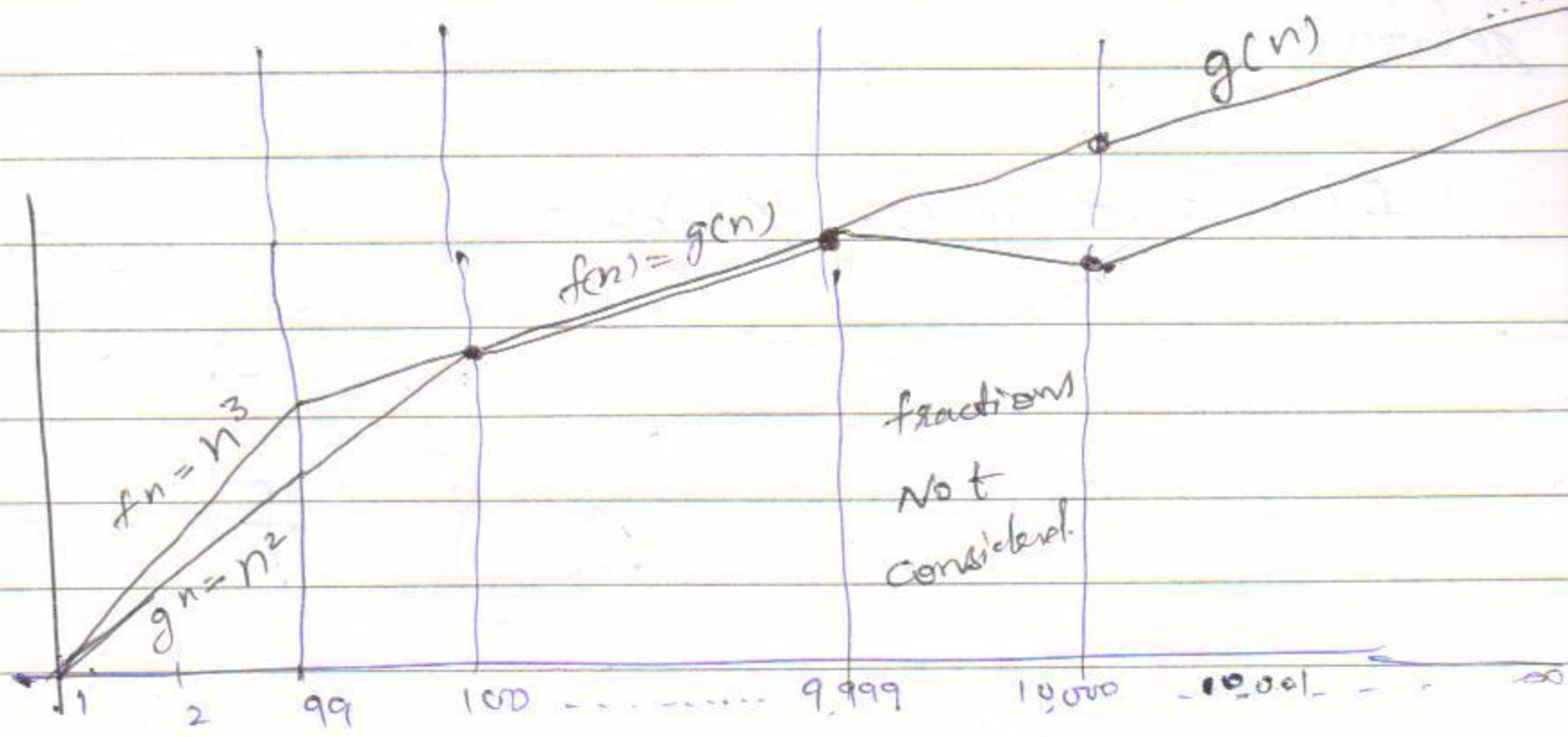
$n \leq C n^3$ $10,000,0000$

19-10-12

Date

$$n = 0 \cdot n^3$$

$$10,000 = C \cdot 1000000$$



only g(n) is always greater or equal ($n_0 = 100$) so only one condition is true

~~f(n) = 0 g(n)~~

$f(n) = 0\ g(n)$

if we change: $f(n) = n^3$

$= n^3$

~~g(n) =~~ .

from here [100]

$n = 65,536$

Q. $f(n) = \log^*(\log n)$ ⇒ 4 times (4 times) Date ____ base 2

$g(n) = \log(\log^* n)$ ⇒ 2 times when base 2 (5 times)

relation.

applying log at the end gives small value

so

$f(n) = O(g(n))$

As

$f(n)$ = divide n by 2

$= n/2$

$(n = 2^{65,536})$

How many times apply log.

65,536

① log 2

② log 65,536

③ log 2^{16} = log 2^4

④ log 4

2

⑤ log 2

1

5 times.

⇒ n ⇒ log (n!) is lower.

1

log ⇒ (log n)! - is greater

⇒ Applying log at the end gives smallest value.

Q. Consider the following 2 functions Date

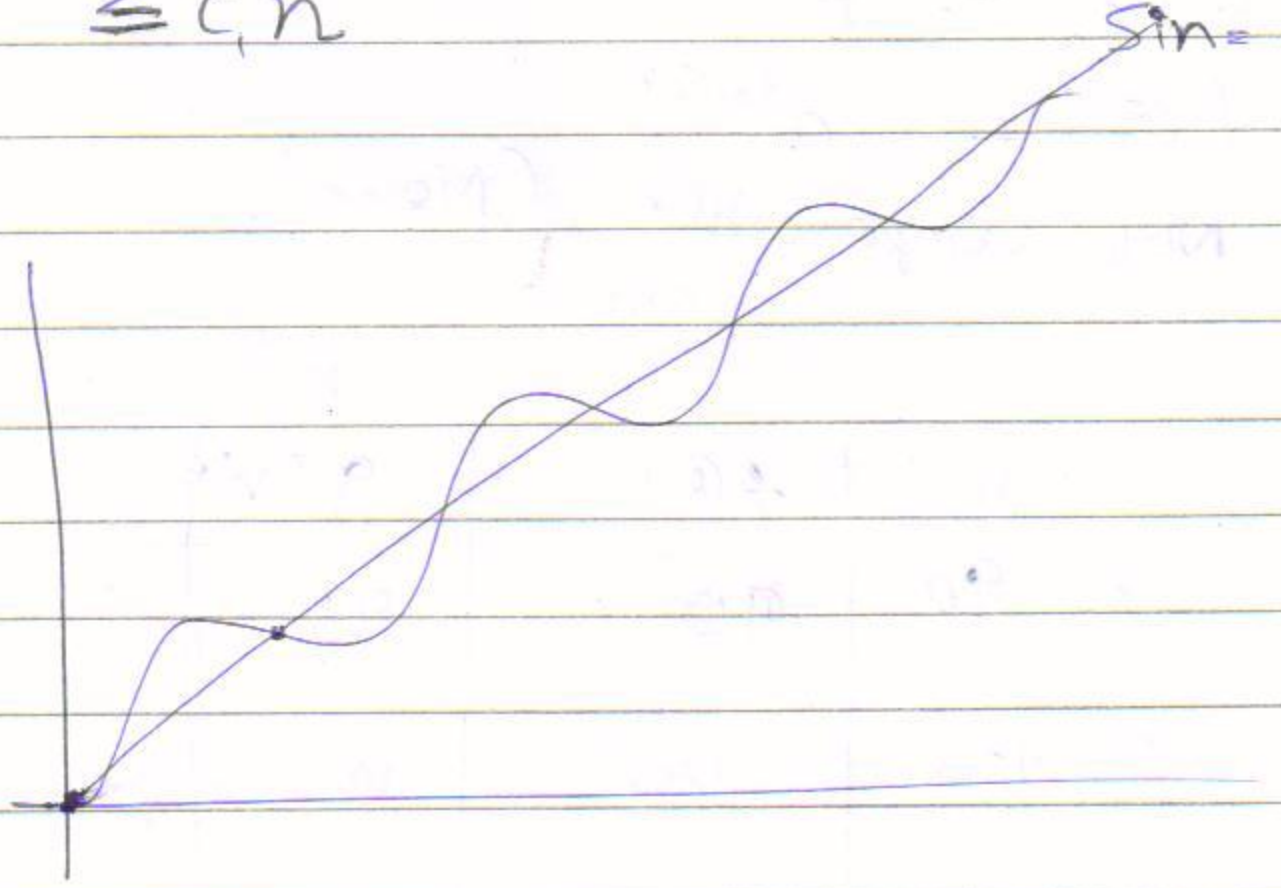
$f(n) = n^{1+\sin(n)}$

$g(n) = n$

what is the relation b/w them ?

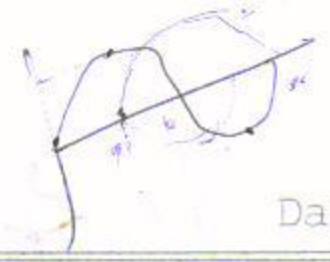
(a) $f(n) = O(g(n))$
(b) $f(n) = \Omega(g(n))$
(c) $f(n) = \Theta(g(n))$
(d) None (Not comparable

Ans. $\because n^{1+\sin(n)} \leq c_1 n$



n	f(n)	g(n)	
n= 90	90^2	90	
180	180	180	=
270	$(270)^{1-1}$	270	
	$(270)^0 = 1$		
360	360	360	

Ans. None



Date

Q. $f(n) = n^{1+\sin n}$

$f(g) = n^2$

n	$f(n)$	$g(n)$	
90	90^2	90^2	✗
180	180^1	180^2	✓
270	$270^{0=1}$	270^2	✓
360	360	360^2	✓

Ans. $f(n) = O\,g(n)$

Q. $f(n) = n^{\sin n}$

$f(g) = n^{\cos n}$

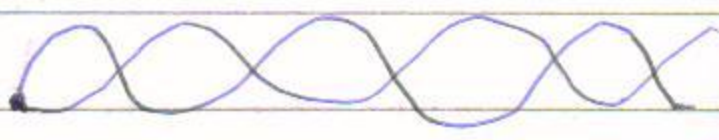
Ans Not comparable (None)

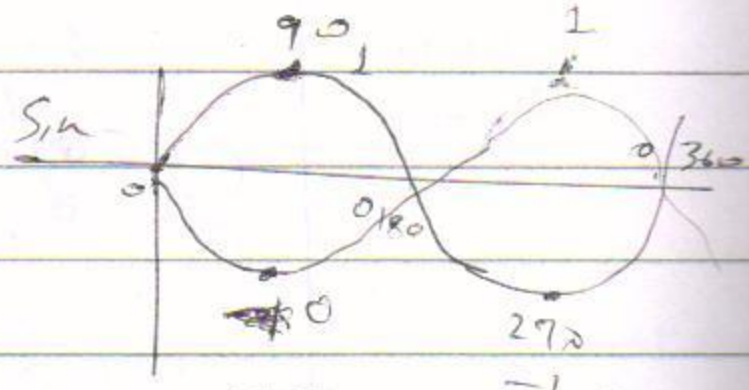
$n^{\sin}$ $n^{\cos}$

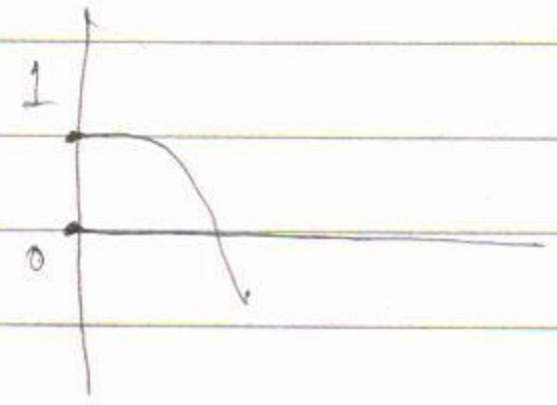
n	$f(n)$	$g(n)$	
✓ 90	$90^1 = 90$	90^0	$=1$
✓ 180	$180^0 = 1$	180^{-1}	$= \frac{1}{180}$
270	$270^{-1} = \frac{1}{270}$	270^0	$=1$ ✓
360	$360^0 = 1$	360^1	$= 360$ ✓

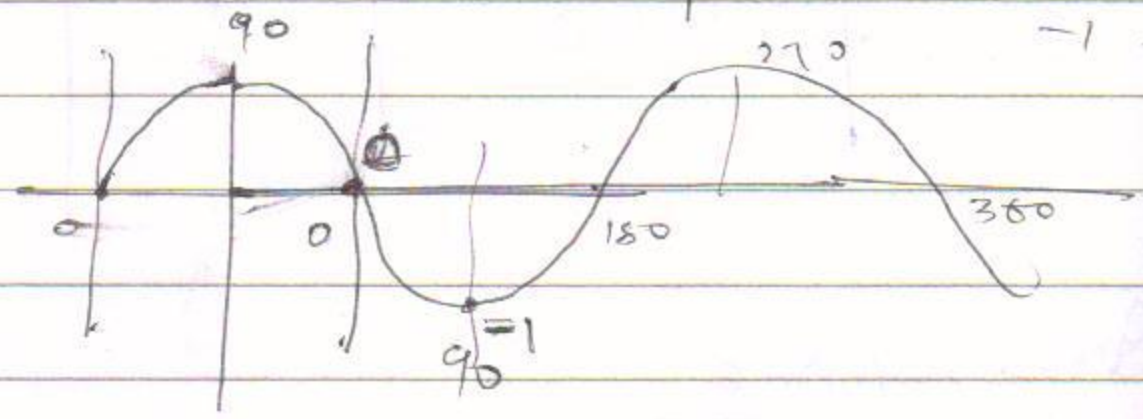
$\sin\theta + \cos\theta \neq 1$

$180^1 = 180$









⇒ Properties of Asymptotic Notation :— Date

(1) Reflexive Property :—

$f(n) = O(f(n))$

$f(n) = \Omega(f(n))$

$f(n) = \Theta(f(n))$

$f(n) \neq o(f(n))$

$f(n) \neq \omega(f(n))$

(2) Symmetric property :—

(1) if, $f(n) = O(g(n))$

then

$g(n) \neq O\ f(n)$ → $n^3 = O n^4$

$n^4 \neq O\ n^3$

(2) if, $f(n) = \Omega(g(n))$

then

$g(n) \neq \Omega(f(n))$ → $n^4 = \Omega(n^3)$

$n^3 \neq \Omega(n^4)$

(3) if, $f(n) = \Theta\ g(n)$

then

$g(n) = \Theta(f(n))$ ✓

(4) If, $f(n) = o(g(n))$ then

$g(n) \neq o\, f(n)$

(5) if $f(n) = \omega(g(n))$ then

$g(n) \neq \omega(f(n))$

3.) Transitivity Property.

If, $f(n) = O(g(n))$ & $g(n) = O(h(n))$ then

(1) $f(n) = O(h(n))$

(2) Similarly, —2—, Θ & o, ω will satisfy Transative property.

(4) If $f(n) = O(g(n))$ then should be a positive function.

$h(n) \cdot f(n) = O(h(n) \cdot g(n))$

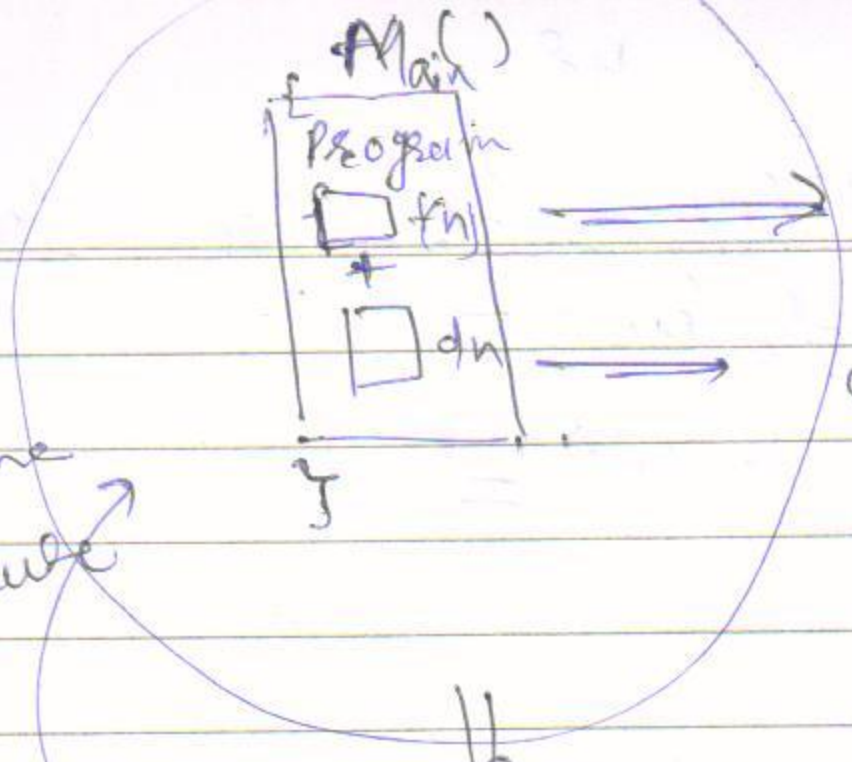
$n^2 = n^2$

$n \cdot n^2 = n^2 \cdot n^2$

(5) If $f(n) = O(g(n))$ && $d(n) = O(h(n))$ then

(1) ~~$f(n) + d(n) = O(g(n) \cdot h(n))$~~ ✓

(2) $f(n) \cdot d(n) = O(g(n) \cdot h(n))$



f(n) max take n^3 in worst case.

$f(n) = O(n^3)$

$d(n) = O(n^5)$

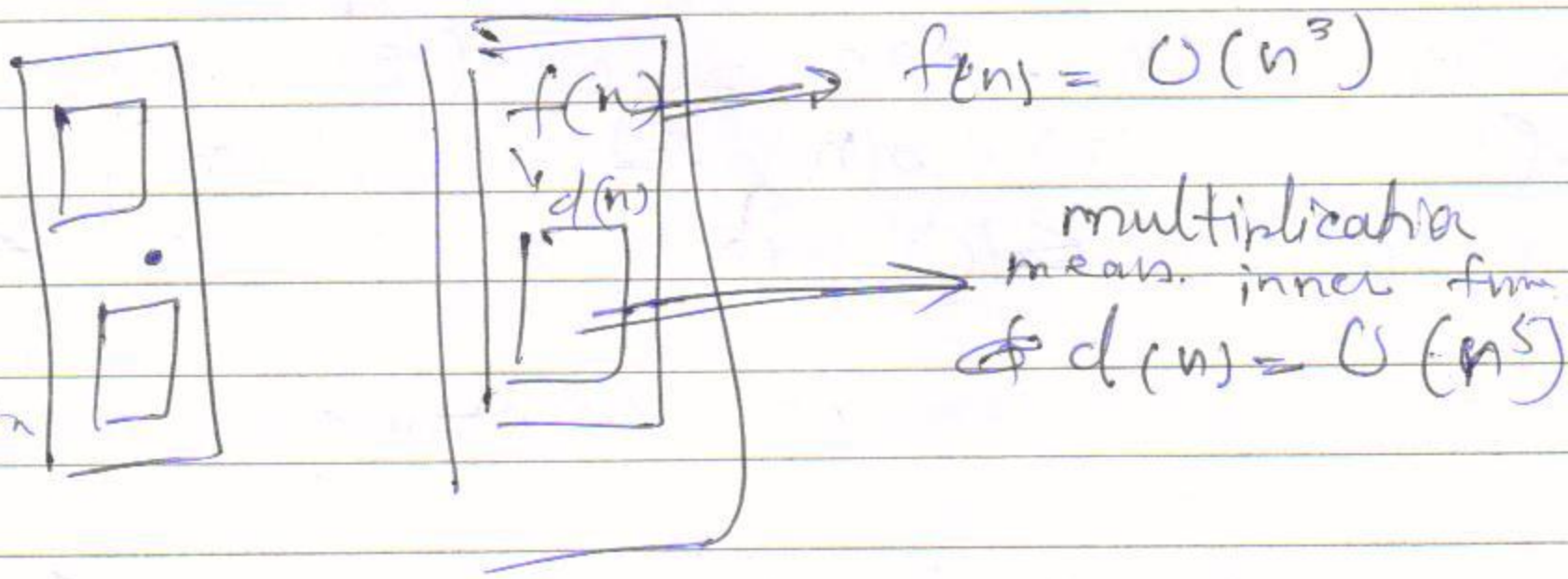
↳ take n^5 in worst case.

$\max(n^3, n^5)$

$\Rightarrow O(n^5)$

① $f(n) + d(n) = \max(g(n), h(n))$

② $f(n) + d(n) = O(g(n) + h(n))$ ✓



✓

② $f(n) \cdot d(n) = O(g(n) \cdot h(n))$

Q. Let $f(n)$, $g(n)$ & $h(n)$, be three functions which are defined as follows -

A $f(n) = O(g(n))$ && $g(n) \neq O(f(n))$

&&

$g(n) = O(h(n))$ && $h(n) = O(g(n))$

T/F ?

(a) $f(n) = O(h(n))$ ✓ true

(b) $f(n) \cdot h(n) = O(h(n) \cdot h(n))$ true

(c) $h(n) = O(f(n))$ false

(d) $h(n) \cdot (g(n)) = O(g(n) \cdot g(n))$ True

(e) $f(n) + g(n) = O(h(n))$ True

A. (a) $f(n) \leq O\ g(n)$ && $g(n) \neq O\ f(n)$

$f(n) < g(n) \rightarrow O$ $f(n) \neq \Omega g(n)$ a $g(n)$ not small.

$g(n) \leq h(n)$ && $h(n) \leq g(n)$

always $g(n) = h(n)$

conclusion ⇒ $f(n) <$ $g(n) = h(n)$

(b) $f(n) \cdot h(n) =$

(e) $f(n) + g(n) = O(h(n))$ let n^2 same $= n^2$

$(n + n^2) = O(n^2)$ so $n^2 = O(n^2)$ ✓ (T) Ans

Changes = consider largest power

$h(n) \neq O(g(n))$

$f(n) < g(n) < h(n)$

a ✓

b ✓

c ✗

d ✗

e ✓

Change in previous problem: Date

If $f(n) \neq O(g(n))$ && $g(n) = O(f(n))$

then (a) × (b) × (c) ✓

(d) ✓ (e) ×

Gate

Q. Suppose $T_1(n) = O(f(n))$ && $T_2(n) = O(f(n))$ then

T/F ?

(a) $T_1(n) + T_2(n) = O(f(n))$ T

(b) × $T_1(n) = O(T_2(n))$ — ~~always true~~ No F.

(c) × $T_1(n) = \Omega(T_2(n))$ — ~~always true~~ F. Can't say

(d) × $T_1(n) = \Theta(T_2(n))$ ~~always true~~ F ~~Can't say~~

d. $T_1(n) \leq c_1 f(n)$ && $T_2(n) \leq c_1(f(n))$

~~$T_1 = f(n)$~~ $T_2 = f(n)$

$T_1 \; T_2 \leq fn$

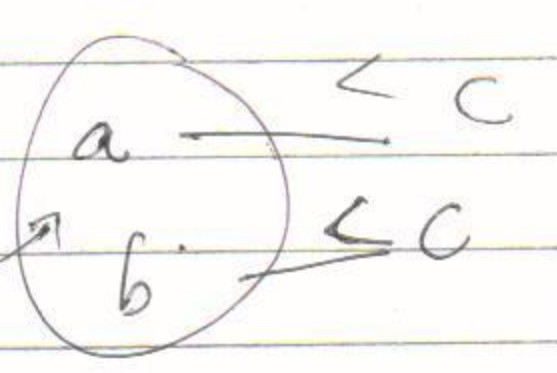
~~if~~ $T_1 = T_2 = T_3$ T_1 && $< fn$

(a) n^2 or n^4 n^5

(i) $T_1(n) = O(f(n))$

n^4 or n^2

$T_2(n) = O(f(n))$ n^5



No relation b/w a & b given

so

If we change

50

Date

$T_1(n) = \theta(f(n))$

$T_2 n = \theta(fn)$

Ans all are there.

Q. If we change

$T_1(n) = \Omega(f(n))$

$T_2(n) = \Omega(fn)$

(a) ✓

(b) there is one case where it is false.

(c) " " " " false

(d) " " " " false.

***
IMP

Ego = 1 / Knowledge

Date 51

⇒ Divide and Conquer :-

1. Recursion
2. Recurrence Relation
3. Recurrence Relation Solving.

(1) Recussion :-

A function is calling that itself to solve a particular problem is called recursion.

= factorial

$f(5) = 5 * f(4) *$

$4 * f(3)$

$3 * f(2)$

$2 * f(1)$

1. ← Stop. forcefully, termination condition.

→ means solving bigger problem in terms of smaller terms.

Requirment : OR When we can solve any problem by recursion →

When every time we need same logic again and again then we can solve that problem using recursion.

⇒ Properties of Recursion :-

1. It should have termination condition

↳ [otherwise program will go into ∞ (infinite loop) then stack overflow.]

it not stuck ∞

stuck overflow ← otherwise ∞ loop

```
A()
{
printf("hi");
}A()
∞
```

(2). It should have recursive steps. Date

[means it will call itself. ⇒ function calling in function]

recursion

A(n)
{
A(n)
}

No Recursion

A( )
{
No parameter passing
A( )
}

(3.) from one function call to another function call parameter value should be changed. { if not changed then again in infinite loop } - bcoz termination condition

want to execute [code] 100 time

termination condition should be before function calling. That's why it is called inicial condition.

[Termination condition = Inicial condition]

c=0

A(c) 0, 1, 2, 3, 4, ... 99, 100
{
if (c==100), return;
Code
c++
A(c)
}

condition depend on value change

★→ Recursive Programming is very good for programmer but not for CPU or Processor and memory bcoz in every function calling it will allocate memory for local variable and other variables and,

Q. Write a recursion program to find out factorial of n.

Store previous local variables and return address into stack till termination condition and then start calculating or evaluating by performing operations on stack.

Sol^{n}:- $f(5) = 5 * f(4)$

$4 * f(3)$

$3 * f(2)$

$2 * f(1)$

1

$$f(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ n * f(n-1) & \text{if } n > 1 \end{cases}$$

$5 = 5 * f(4)$

$120 = 5 * f(4)^{24}$

$4 * f(3)^{6}$

$3 * f(2)^{2}$

$2 * f(1)$

1

⇒ Recursive Program:-

Date

```
fact (n)
{
   if (n <= 1) return n(1);
      else
        {
          return ( n * fact (n-1) );
        }
}
```

Time complexity of Recursive program.
How many times function is repeating or calling.
Here in fact (n)

$$TC = O(n)$$

Q. Write a recurrance relation and ~~write~~ recursive program, to find multiplication of 2 potive integer.

Ans.

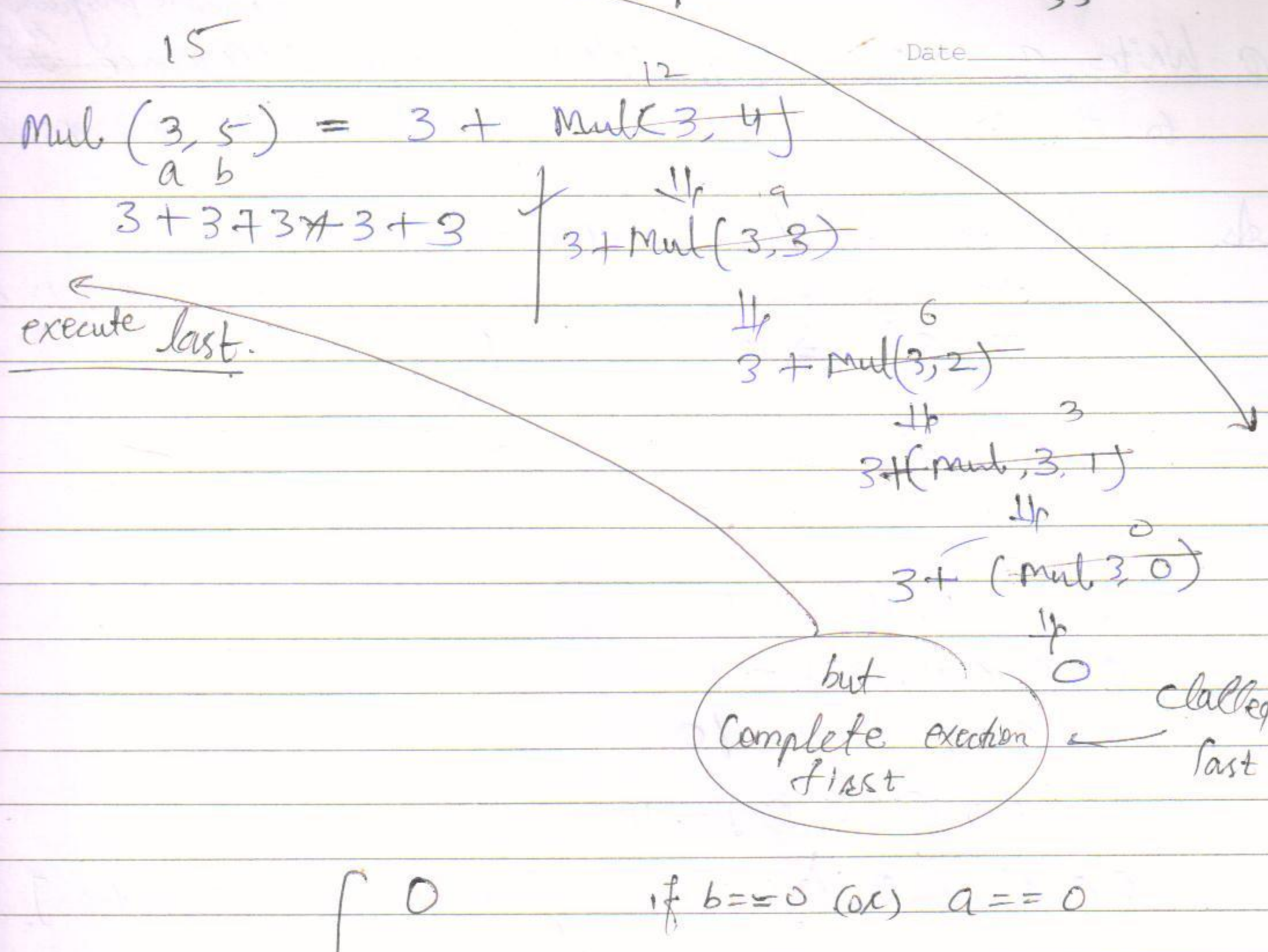
$$f(n) = \begin{cases} \\ \\ \end{cases}$$

a * b b
10 = 2 * 5 *
= 10
a * b

a + a
2 + 2 + 2 + 2 + 2 =
(a + a)
+
a + a
└ a + a
└ a + a

if (a =

Date



$$Mul(a,b) = \begin{cases} 0 & \text{if } b==0 \text{ (or) } a==0 \\ a + Mul(a, b-1) & \text{otherwise.} \end{cases}$$

Time complexity:

$$T(Mul(a,b)) = b+1 \Rightarrow O(b)$$

Recursive Program:-

```
Mul (a, b)
{
  if (a==0 || b==0) return (0);
  else
  {
    return (a + Mul (a, b-1))
  }
}
```

Q. Write a Recurrance relation and recursive program to find GCD of two positive numbers m & n.

A. GCD → Greatest common Divisor

m & n ∈ 1, 2, ...

GCD (5, 3) ⇒ prime no.s. = 1

5 = {1, 2, ...

GCD (10, 100) = 10

GCD (5, 13) = 1

GCD (21, 23) = 1

GCD (0, 10) = 10

GCD (100, 0) = 100

GCD (0, 0) = ∞ [undefined condition].

GCD (21, 23)

⇓

21) 23 (1
21
---
2) 21 (10
20
---
1) 2 (2
2

0

(1) ←

GCD (21, 23)

⇓

GCD (21, 23)

21] 23] 1
21

2

GCD (2, 21)

2) 21] 10
20

1

GCD (1, 2)

1) 2 (2
2

0

GCD (0, 1) = 1

Date

m n

GCD (9, 23)

9) 23 (2
18

5

GCD (5, 9)

5) 9 (1
5

4

GCD (4, 5) ←

4) 5 (1
4

1

GCD (1, 4)

1) 4 (4
4

0

n m (n%m, m)

23 % 9 ⇒ 5, 9

GCD (0, 1) = 1

$$GCD(m,n) = \begin{cases} \infty & \text{if } m==0 \&\& n==0 \\ n & \text{if } m==0 \\ m & \text{if } n==0 \\ GCD(n\%m, m) & \text{otherwise.} \end{cases}$$

GCD(m, n)
9 (23)

largest

log 23 + 1
4

log 2 + 1

4 x log 2.

4 times + 1

5 times

$\log_2 n$

Worst case ⇒

$T(GCD(m,n)) = O(\log_2 n)$ if $m \le n$

best case ⇒

$T(GCD(m,n)) = \Omega(1)$.

Average case ⇒.

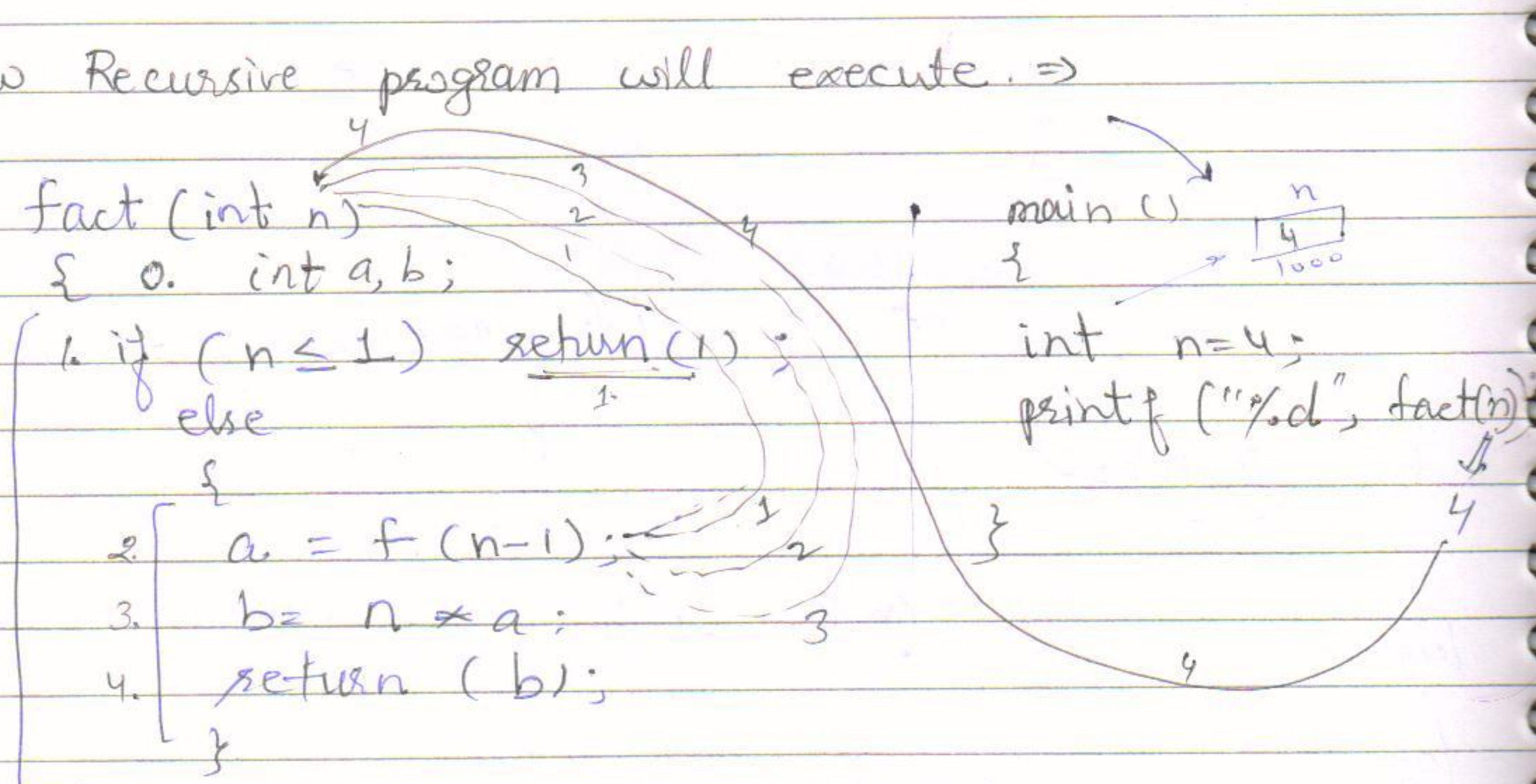
$T(GCD(m,n)) = O(\log n)$

Date

Recursive Program :-

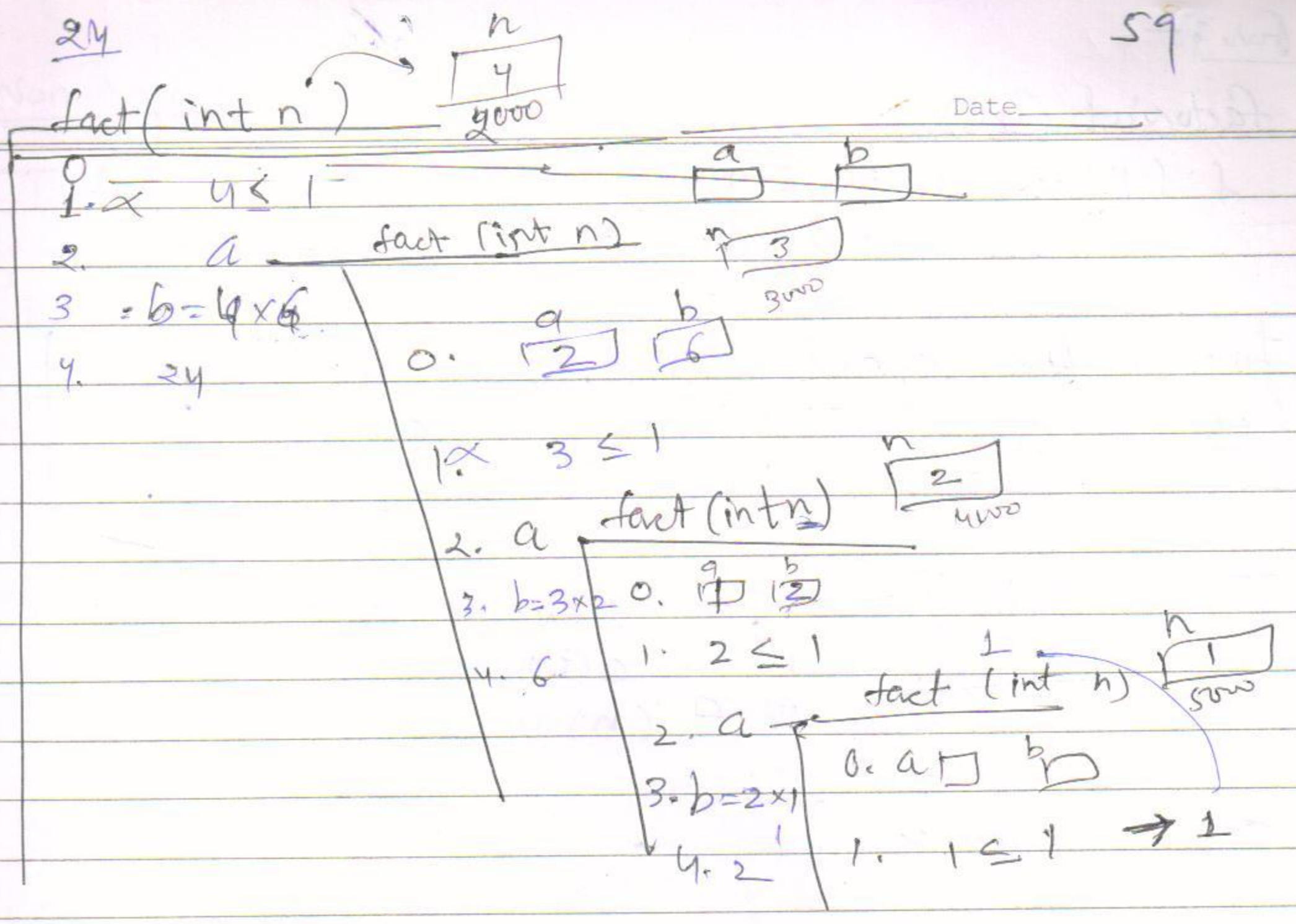
```
GCD (m, n)
{
  if (m==n==0) return (∞)
  else
    if (m==0) return (n)
    else
      if (n==0) return (m)
      else
        return ( GCD (n % m, m) ;
}
```

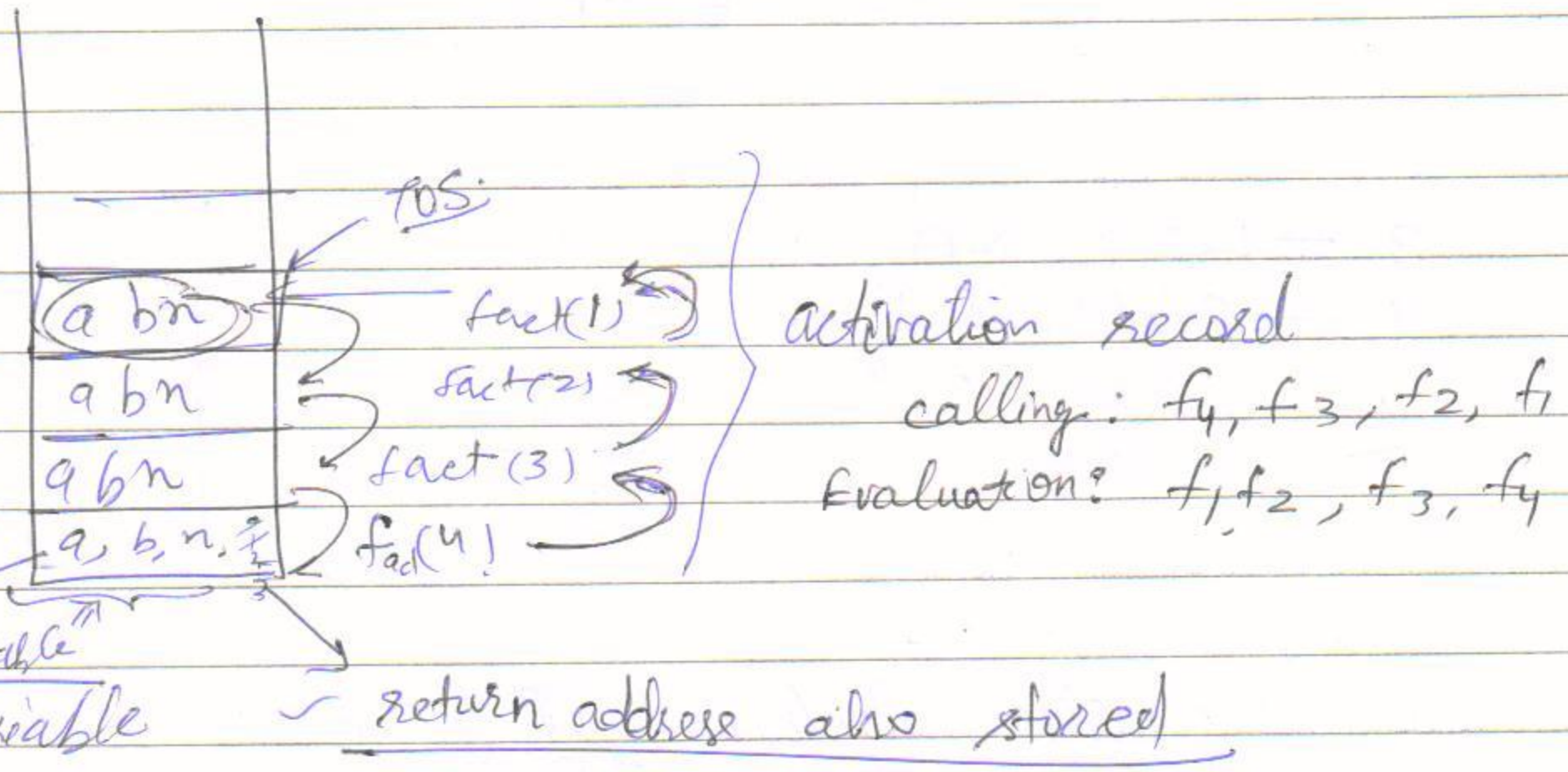
⇒ How Recursive program will execute. ⇒



→ run time binding

24





4 slots of stack is used and every ~~stack~~ slot is contains 3 local variable each having 2 byte = total 6 byte in each

$4 \times 6 = 24$ B.

For :-

factorial (n)

$f(n) = n * 6B$

$= 6.nB$

$= O(n)$

finding the stack size, is the space complexity in recursive program

$T(fact(n)) = n * 1$ multiplication

$= n * O(1)$

$= \Theta(n)$

Space Complexity.

$S(fact(n)) = n * 6B$

$= 6nB$

$= \Theta(n)$

3 — Local V(Int) → 6B (2B)

1 — Global V(Float) 4B → 4B

1 — Return (3 3B → 3B
address

13B

100 times

13

$\phi = 7$

Non - Recursive factorial Program:-

```
fact (int n)        2B
{
    int s=1, i;     2B  2B   6B.
    for (i=1; i<=n; i++)
    {
        s = s * i;
    }
}
```

Non-recursive :→

Date

$T(\text{Fact}(n)) = O(n)$

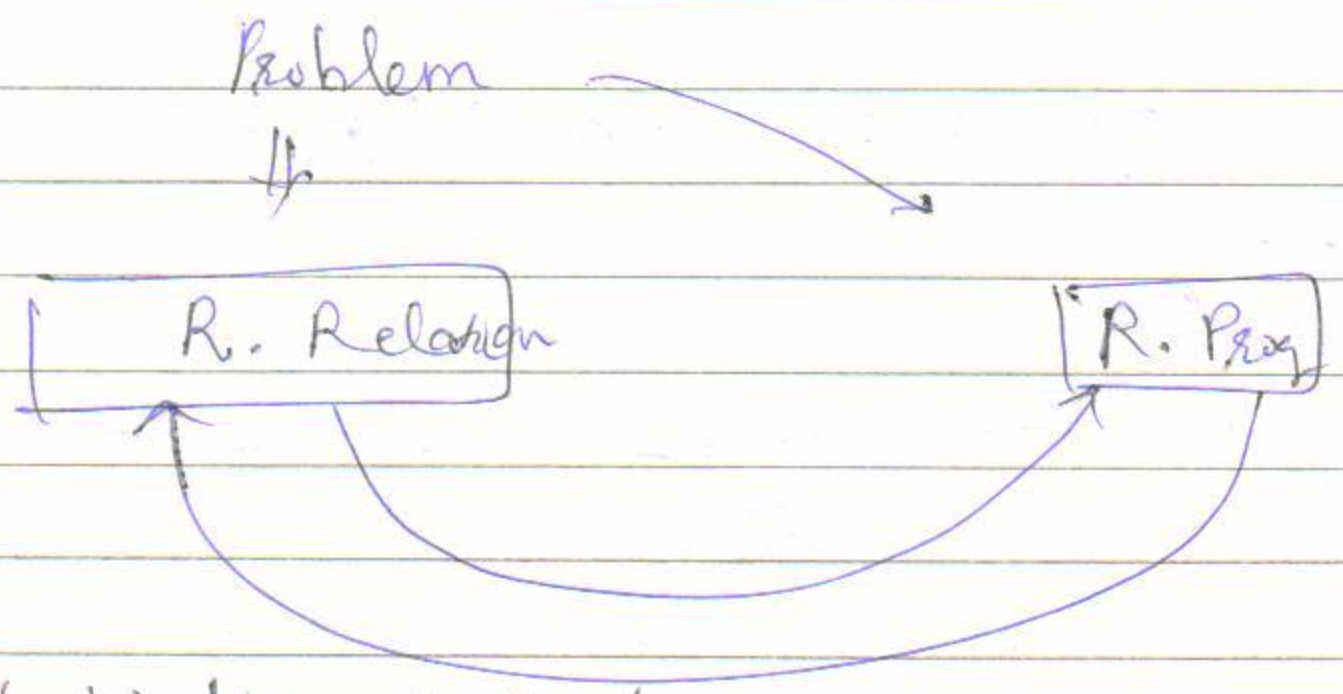
Space Complexity

$S(\text{Fact}(n)) = O(1)$

(1) The data structure we use in recursion is stack.
(2) Comparing Recursive & Non-recursive, program, recursive program will take more space.
(3) Imp.

⇒ Recurrence Relation solving :-

(1) Substitution method.
(2) Recursive Tree method.
(3) Master Theorm.



(i) Substitution Method :-

Substituting the given function again and again untill given function is removed, is called substitution method.

Date

eg-1:-

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n-1)+n & \text{if } n>1 \end{cases}$$

$T(n) = T(n-1) + n$

$T(50) = T(50-1) + 50$

$= T(49) + 50$

$T(2) = T(1) + 2$

$T\left(\frac{n}{2}\right) = T\left(\frac{n}{2}-1\right) + \frac{n}{2}$

$T(n) = T(n-1) + (n-0)$ ① time

$= T(n-2) + (n-1) + (n-0)$ ② time

$= T(n-3) + (n-2) + (n-1) + (n-0)$ ③ time

$= T(n-50) + (n-49) + (n-48) \cdots + (n-1) + n$

$= T(n-(n-1)) + (n-(n-2)) + (n-(n-3)) + \cdots (n-1) + n$

$= T(1) + 2 + 3 + 4 - - - - + (n-1)$

T.C. $= 1 + 2 + 3 + 4 \cdots n = \frac{n(n+1)}{2} = O(n^2)$

Q. $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ [T(n-1)] * n & \text{if } n > 1 \end{cases}$

$= T(n-1) * n$ — (1)

$= T(n-1-1) * (n-1) * n$ — (2)

$= T(n-2) * (n-1) * n$

$= T(n-1-2) * (n-1) * n$ — (3)

$= T(n-3) * (n-2) * (n-1) * (n-0)$

$= T(n-(n-1)) * (n-(n-2)) * (n-(n-3)) * \cdots * (n-1) * (n-0)$

$n - k = 1$
$k = n-1$

$= 1 * 2 * 3 * 4 * \cdots * (n-1) * n$

$= n!$

upper bound of $n! = O(n^n)$ — (3)

Q. $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + \log(n) & \text{if } n > 1 \end{cases}$

Ans. $= T(n-1) + \log(n)$

$= \cancel{T(n-1) + \log(n) + \log n}$

$= T(n-1-1) + \log(n-1) + \log n$

$= T(n-2) + \log(n-1) + \log n$

$= T(n-3) + \log(n-2) + \log(n-1) + \log(n-0)$

$\} (n-1)$

$T = T(n-(n-1)) + \log(n-(n-2)) + \log(n-(n-3)) + \cdots$

$= 1 + \log 2 + \log 3 + \log 4 + \cdots + \log n$

$= 1 + \log (1 \cdot 2 \cdot 3 \cdot 4 \cdots n)$

$= 1 + \log (n!)$

$= 1 + \log (n^n)$ → upper bound

$= 1 + n \log n$

$= \boxed{T(n \log n)}$

Q. $T(n) \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + \frac{1}{n} & \text{if } n > 1 \end{cases}$

$= T_n$

$= T(n-1) + \frac{1}{n}$

$= T(n-2) + \frac{1}{n-1} + \frac{1}{n}$

$= T(n-3) + \frac{1}{(n-2)} + \frac{1}{(n-1)} + \frac{1}{n}$

$n-1$

$= T(n-(n-1)) + \frac{1}{n-(n-2)} + \frac{1}{n-(n-3)} + \cdots + \frac{1}{n}$

$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}$

$= O(\log n)$

$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4}$

$= 1$
0.5
0.33
0.25
$2.08 \ldots$

Q. $$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ T(n-2) + \log(n) & \text{if } n > 0 \end{cases}$$

65

assume n value is even

Date

Ans: $T_n = T(n-2) + \log(n)$ (2×1)

① $= T(n-2-2) + \log(n-2) + \log n$

$= T(n-4) + \log(n-2) + \log(n-0)$ ($C_{2\times 2}$)

② $= T(n-6) + \log(n-4) + \log(n-2) + \log n$ (2×3)

③ $= T(n-8) + \log(n-6) + \log(n-4) + \log(n-2) + \log n$ (2×4)

$n-8 = 0$ (2×k) (2(0)−2) (2k−2)

$k = n$ $\quad n-2k = 0$ $\quad n = 2k$ $\quad \frac{n}{2} = k$

$T(n-n) + \log(n-(n+2)) +$

$= T(n-2k) + \log(n-(2k-2)) + \log(n-(2k-4)) + \cdots + \log(2k-0)$

$n = 2k$

$= T(2k-2k) + \log(2k-(2k-2)) + \log(2k-(2k-4)) + \cdots + \log(2k-0)$

$= T(0) + \log 2 + \log 4 + \log 6 + \cdots + \log 2k$

$= 1 + \log(2\times 1) + \log(2\times 2) + \log(2\times 3) + \cdots + \log(2\times k)$

$= 1 + (\log 2 + \log 1) + (\log 2 + \log 2) + (\log 2 + \log 3) + (\log 2 + \log 4) + \cdots + (\log 2 + \log k)$

$= 1 + (k) + \log_2^2 + [\log 1 + \log 2 + \log 3 + \cdots + \log k]$

$= k + \log(k!) \Rightarrow k + \log(k^k) \Rightarrow k + k\log k$

$\Rightarrow \frac{n}{2} + \frac{n}{2}\log\frac{n}{2} \Rightarrow O(n \log n)$ Ans

Q. $T_n = \begin{cases} 1 & \text{if } n=0 \\ T(n-2)+n^2 & \text{if } n>0 \end{cases}$

Date

Q. $T(n) = \underset{2\times1}{T(n-2)} + n^2$

$\hookrightarrow = \underset{2\times2}{T(n-4)} + (n-2)^2 + n^2$

$\hookrightarrow = \underset{2\times3}{T(n-6)} + (n-4)^2 + (n-2)^2 + (n-0)^2$

$\hookrightarrow = T(n-2k) + (n-(2k-2))^2 + (n-(2k-4))^2 + \cdots + (n-0)^2$

$n-2k=0$

$2k=n$

$= T(n-2k) + (n-(2k-2))^2 + (n-(2k-4))^2 + \cdots + (n-0)^2$

$= T(n-2k) + (2k-(2k-2))^2 + (2k-$

$= T(0) + 2^2 + 4^2 + 6^2 + 8^2 + \cdots + (2\cdot k)^2$

$= 1 + (2\cdot1)^2 + (2\cdot2)^2 + (2\cdot3)^2 + (2\cdot4)^2 + \cdots + (2k)^2$

$= 1 + 2^2[1^2 + 2^2 + 3^2 + 4^2 + \cdots k^2]$

$= 1 + \frac{k(k+1)(2k+1)}{6} = k^3$

$\Rightarrow \left(\frac{n}{2}\right)^3 \Rightarrow \frac{1}{8} \cdot n^3 \Rightarrow O(n^3)$

Date

① $T(n) = T(n/2) + n_0$, $T(1) = 1$

② $T(n) = 2T(n/2) + n_0$, $T(1) = 1$

③ $T(n) = 2T(n/2) + n\log n_0$, $T(1) = 1$

④ $T(n) = 2T(n/2) + \frac{n}{\log n}$, $T(1) = 1$

⑤ $T(n) = 8T(n/2) + n^2$, $T(1) = 1$

⑥ $T(n) = 2T(n-1) + n$, $T(1) = 1$

⑦ $T(n) = \sqrt{n}\,T(\sqrt{n}) + n$, $T(2) = 2$

Q.1) $T(n) = T(n/2) + n_0$, $T(1) = 1$

A) $T(n) = T(n/2) + n$

$= T\left(\frac{n}{2.2}\right) + \left(\frac{n}{2}\right) + \frac{n}{2^0}$

$= T\left(\frac{n}{2.2.2}\right) + \left(\frac{n}{2.2}\right) + \left(\frac{n}{2}\right) + \frac{n}{2^0}$

$= T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} + \frac{n}{2^1} + \frac{n}{2^0}$

$\vdots$

$+ 2^{(k)}$ putting k

$= T\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}} + \frac{n}{2^{k-2}} + \cdots + \frac{n}{2^1} + \frac{n}{2^0}$

$a \cdot ar \cdot ar^2$ ... $(a \cdot r^k)^{\log_2 n}$ $2^k = 2^{\log_2 n}$ ⇒ value less than 1

$2^4 \; 2^3 \; 2^2$ ... $\frac{(a \cdot r^k)}{1-r}$ decreasing series

(68)

Date

$$= T\left(\frac{n}{2^{\log_2 n}}\right) + \frac{n}{2^{\log_2 n - 1}} + \frac{n}{2^{\log_2 n - 2}} + \cdots + \frac{n}{2^1} + \frac{n}{2^0}$$

$n^{\log_2 1}$ $\frac{n^{\log_2 2}}{2^1}$ $\frac{n^{\log_2 n}}{2^2}$ $+ \frac{n^{\log_2 3}}{2^3} + \cdots$

$$T(n) = T(1) + n\left[\left(\frac{1}{2}\right)^0 + \left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^2 + \cdots + \left(\frac{1}{2}\right)^{\log_2 n - 1}\right]$$

$$= 1 + n\left[\frac{1\left(1 - \left(\frac{1}{2}\right)^{\log n}\right)}{1 - \frac{1}{2}}\right]$$

$\frac{a(1-r^n)}{1-r}$

$r \ll 1$

$r > 1$

$\Downarrow$

$\frac{a(r^n - 1)}{r-1}$

$$= 1 + n\left[1 - \frac{1}{2^{\log_2 n}}\right]$$

$$= 1 + n\left[1 - \frac{1}{n}\right]$$

$$= 1 + n[1 - 0]$$

$a + ra + ar^2 + \cdots$

$\frac{ar}{a} = r$

$$= O(n)$$

Q. $T(n) = 2T(n/2) + n$, $T(1) = 1$

Ans. $T(n) = 2T(n/2) + n$

$= 2T$

$T(50) = 2 \times T(50/2) + 50$

$T(4)^2 = 2 \times T(4/2) + 2$

$T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n$

$= 2^2 T\left(\frac{n}{2^2}\right) + \frac{n}{2} + n$

$\hookrightarrow = 2^2\left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right] + n + n$

$= 2^3 T\left(\frac{n}{2^3}\right) + n + n + n$

$\vdots$ k

$= 2^k T\left(\frac{n}{2^k}\right) + kn$

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n$

$= 2^{\log_2 n}(1) + n \cdot \log_2 n$

$= n + n \cdot \log_2 n$

$= O(n \log_2 n)$

Q. $T(n) = 2T(n/2) + n\log n$ Date

$= 2\left[2T(n/2^2) + \frac{n}{2}\log n/2\right] + n\log n$

$= 2\left[2\left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\log\frac{n}{2^2}\right] + \frac{n}{2}\log n/2\right] + \frac{n}{2}\log n$

$= 2^2\left[2T(n/2^3) + \frac{n}{2^2}\log\frac{n}{2^2}\right] + n\log\frac{n}{2^1} + n\log\frac{n}{2^0}$

$= 2^3 T(n/2^3) + n\log\frac{n}{2^2} + n\log\frac{n}{2^1} + n\log\frac{n}{2^0}$

$\downarrow k$

$= 2^k T\left(\frac{n}{2^k}\right) + n\left[\log\frac{n}{2^0} + \log\frac{n}{2^1} + \log\frac{n}{2^2} + \cdots + \log\frac{n}{2^{k-1}}\right]$

write in reverse order

$T_n = nT(1) + n\left[(\log n - 0) + (\log n - 1) + (\log - 2) + (\log n - 3) + \cdots + \log n(\log n)\right]$

$= n * 1 + n\left[1 + 2 + 3 + 4 + \cdots + \log n\right]$

$= n + n\left[\frac{\log n\,(\log n + 1)}{2^2}\right]$

$= n + n(\log n)^2$

$= O\left(n(\log n)^2\right)$ Ans.

Q. $T(n) = 2T(n/2) + \frac{n}{\log n}$ → Data solve and get same

$= n + n\left\{\frac{1}{\log \frac{n}{2^0}} + \frac{1}{\log \frac{n}{2^1}} + \frac{1}{\log \frac{n}{2^2}} + \cdots + \frac{1}{\log \frac{n}{n^{\log n}}}\right.$

$\left. + \frac{1}{\log \frac{n}{2^{\log n - 1}}}\right\}$

$= n + n\left[\frac{1}{(\log n - 0)} + \frac{1}{(\log n - 1)} + \frac{1}{(\log n - 2)} + \cdots + \frac{1}{(\log n - \log n - 1)}\right]$

$= n + n\left\{\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{\log n}\right\}$

$= n + n\left[\log(\log^n)\right]$

$= O(n \log \log n)$ Ans.

Q. $T(n) = 8T(n/2) + n^2$

$= 8\left[8T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2}\right)^2\right] + n^2$

$= 8^2 T\left(\frac{n}{2^2}\right) + 2n^2 + n^2$

$= 8^2\left[8T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2\right] + 2n^2 + n^2$

$= 8^3 T\left(\frac{n}{2^3}\right) + 2^2 n^2 + 2^1 2^2 + 2^0 n^2$

$\vdots$

k

$\frac{n}{2^k} = 1$ ∴ $n = 2^k$

$k = \log_2 n$ TL

Date

$= 8^k T\left(\frac{n}{2^k}\right) + n^2 \left[2^0 + 2^1 + 2^2 + \cdots + 2^{k-1}\right]$

$= 8^{\log_2 n} T(1) + n^2 \left[2^0 + 2^1 + 2^2 + \cdots + 2^{\log n - 1}\right]$

$= n^3 \cdot 1 + n^2 \left[1 \cdot \frac{(2^{\log_2 n} - 1)}{2-1}\right]$

$= n^3 + n^2 \left[\frac{n-1}{1}\right]$

$= n^3 + n^2 (n-1)$

$= n^3 + n^3 - n^2$

$= 2n^3 - n^2$

$= 2n^3$ — largest term

$= 2n^3$

$= O(n^3)$ Ans

~~$T(n) = 2T(n-1) + n$~~

Q. $T(n) = \sqrt{n}\, T\sqrt{n} + n$, $T(2) = 2$

$= \sqrt{n}\, T(n)^{1/2} + n$

$= n^{1/2} \left[n^{1/2^2} T\left(n^{\frac{1}{2}}\right) + n^{1/2}\right] + n$

$= n^{3/2^2} \left[T\left(n^{1/2^3}\right) + n^{1/2}\right] + n$

$= n^{3/2^2} \left[n^{1/2^3} T\left(n^{1/2^3}\right) + n^{1/2^2}\right] + n + n$

$n^{1/2} \cdot n^{\frac{1}{2^2}} = n^{\frac{1}{2} + \frac{1}{2^2}}$

$= n^{\frac{2+1}{4}}$

$= n^{\frac{3}{2^2}}$

Date

$$= n^{7/2^3} T(n^{1/2^3}) + n + n + n$$

$$= T(n) = n^{\frac{2^k-1}{2^k}} T(n^{1/2^k}) + kn$$

$$= n^{1-\frac{1}{2^k}} T(n^{1/2^k}) + kn$$

$$= \frac{n}{n^{\frac{1}{2^k}}} T(n^{\frac{1}{2^k}}) + kn$$

$$= \frac{n}{n^{\frac{1}{2^{\log\log n}}}}$$

$$= \frac{n}{2} T(2) + n \log\log n$$

$$= n + n \log\log n$$

$$= O(n \log\log n) \quad \text{Ans}$$

T.C.

$n^{1/2^k} = 2$

$\log n^{1/2^k} = \log 2$

$\frac{1}{2^k} \log n = 1$

$\log n = 2^k$

$\log\log n = k \log 2$

$n^{\frac{1}{2^k}} = n^{\frac{1}{2^k}}$

$\frac{1}{n^{2^k}} = \left(2^{2^k}\right)^{\frac{1}{2^k}}$

Q

$T(n) = 2T(n-1) + n$. $T(1) = 1$

Date

$= 2[2T(n-2) + (n-1)] + n$

$= 2^2 T(n-2) + 2^1(n-1) + 2^0(n-0)$

$= 2^2[2T(n-3) + (n-2)] + 2^1(n-1) + 2^0(n-0)$

$= 2^3 T(n-3) + 2^2(n-2) + 2^1(n-1) + 2^0(n-0)$

$= \sum_{k}$ $n-k=1 \Rightarrow k=n-1$

$= 2^{n-1}[(n-(n-1)) + 2^{n-2}(n-(n-2)) + 2^{n-3}(n-(n-3)) +$

$\cdots + 2^1(n-1) + 2^0(n-0)$

$= 2^{n-1} T(1) + 2^{n-2}(2) + 2^{n-3}(3) + \cdots +$

$+ 2^3(n-3) + 2^2(n-2) + 2^1(n-1)$
$+ 2^0(n-0)$

write in reverse order

combination of AP & GP.

$= 2^0(n-0) + 2^1(n-1) + 2^2(n-2) + \cdots$

$+ 2^{n-3} \cdot 3 + 2^{n-2} \cdot 2 + 2^{n-1} \cdot 1$

⇒ Convert into GP. (How to convert AP into G.P) :—

~~$2^1 \cdot 5 + 2^2 \cdot 6 + 2^3 \cdot 7 + 2^4 \cdot 8$~~

⇒ $2^1 \cdot 1 + 2^2 \cdot 2 + 2^3 \cdot 3 + 2^4 \cdot 4$

multiplied by 2 / multiplied by 2 value. subtract both

$$T(n) = 2^0(n-0) + 2^1(n-1) + 2^2(n-2) + \cdots + 2^{n-2}\cdot 2 + 2^{n-1}\cdot 1$$

(2) $2T(n) = \quad +2^1(n-0) + 2^2(n-1) + \cdots + 2^{n-2}\cdot 3 + 2^{n-1}\cdot 2 + 2^n\cdot 1$

subtract (−)

$$T(n) - 2T(n) = 2^0(n-0) - 2^1 - 1\cdot 2^2 - 1\cdot 2^3 - 1\cdot 2^4 \cdots - 1\cdot 2^4 - 1\cdot 2^{n-1} - 1\cdot 2^n$$

$$-T(n) = n - 1\left[2^1 + 2^2 + 2^3 + \cdots + 2^n\right]$$

$$-T(n) = n - \left[2\cdot\frac{(2^n - 1)}{2-1}\right] \Rightarrow$$

$$-T(n) = n - \left[2^{n+1} - 2\right]$$

multiply by (−)

$$+T(n) = -n + 2^{n+1} - 2$$

$$T(n) = 2^{n+1} - 2 - n$$

$$= 2^{n+1}$$

$$T(n) = 2\cdot 2^n \Rightarrow O(2^n)$$ Ans.

$$T(n) = \left(\frac{1}{2}\right)^0 + \left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^2 + \cdots + \left(\frac{1}{2}\right)^n$$ → GP series.

(1, 0.5, 0.25)

$$= 1\left[1 - \right.$$

$$= O(1)$$

$T(n) = (4/7)^0 + (4/7)^1 + (4/7)^2 + \cdots$ 76

Date

$= \{1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}\}$ — not decrease in G.P serese.

$= \log n$

$\Rightarrow T(n) = 2T(n/2) + n$

$= O(n \log n)$

$T(n) = 2T(n/2) + 2n$

$= 2n \log n$

$= O(n \log n)$

$T(n) = 2T(n/2) + \frac{n}{2}$

$= O(n \log n)$

$\Rightarrow T(n) = 2T(n/2) + n$

$= 2T\left(\frac{n}{2} + 10\right) + n$

↳ n change by constant is small value

$\Rightarrow T(n) = 2T\left(\left[\frac{n}{2}\right]\right) + n$

$=$ same $O(n \log n)$ +1 & -1 can not effect the Ans

$\Rightarrow T(n) = 2T\left(\left\lceil\frac{n}{2}\right\rceil\right) + n$

Date

⇒ $T(n) =$ ——— $+ n$

$=$ ——— $+ \frac{n}{2} + n$

$+ n\left(\frac{1}{2} + \frac{1}{2^0}\right)$

↪ decreasing G.P series you can stop.

and

⇒ $T(n) =$ ——— $+ n$

$=$ ——— $+ n + n$

stop.

sum

how many time n.

⇒ Divide and Conquer :-

an array which content 100 enements. $S(100)$

are you small (50) (50)

25 25 25 25

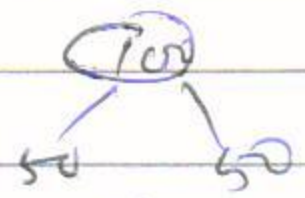
12 12 12 12 12 12 12 12

⇒ My divide algo is best means, divdation and combination is taking very less time.

⇒

Date:

Divide:- Divide the given problem into some sub-problem.



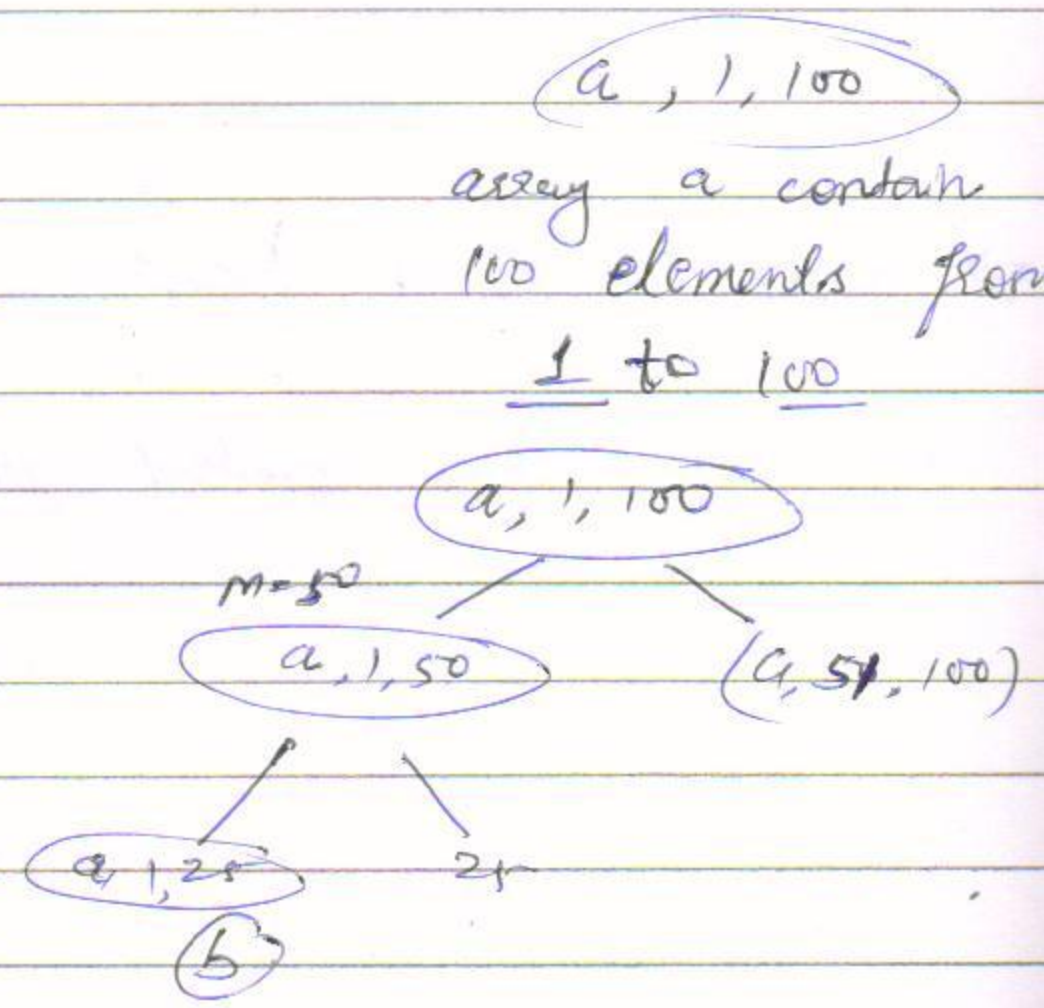
Conquer:- Conquer the sub-problems by calling recursively untill sub problem solutions are available.

Combine:- Combine the subproblem solutions to get original problem solution.

⇒ Control Abstraction of Divide & Conquer:-

```
DAC (a, p, q)
{
  if (small (a, p, q))
    return (a, p, q);
  else
  {
    m = Divide (a, p, q); } divide
    b = DAC (a, p, m);
      = Divide (a, m+1, q); } conquer
    c = DAC (a, m+1, q);
    d = combine (b, c); } combined
    return (d);
  }
}
```

a, 1, 100
array a contain 100 elements from 1 to 100



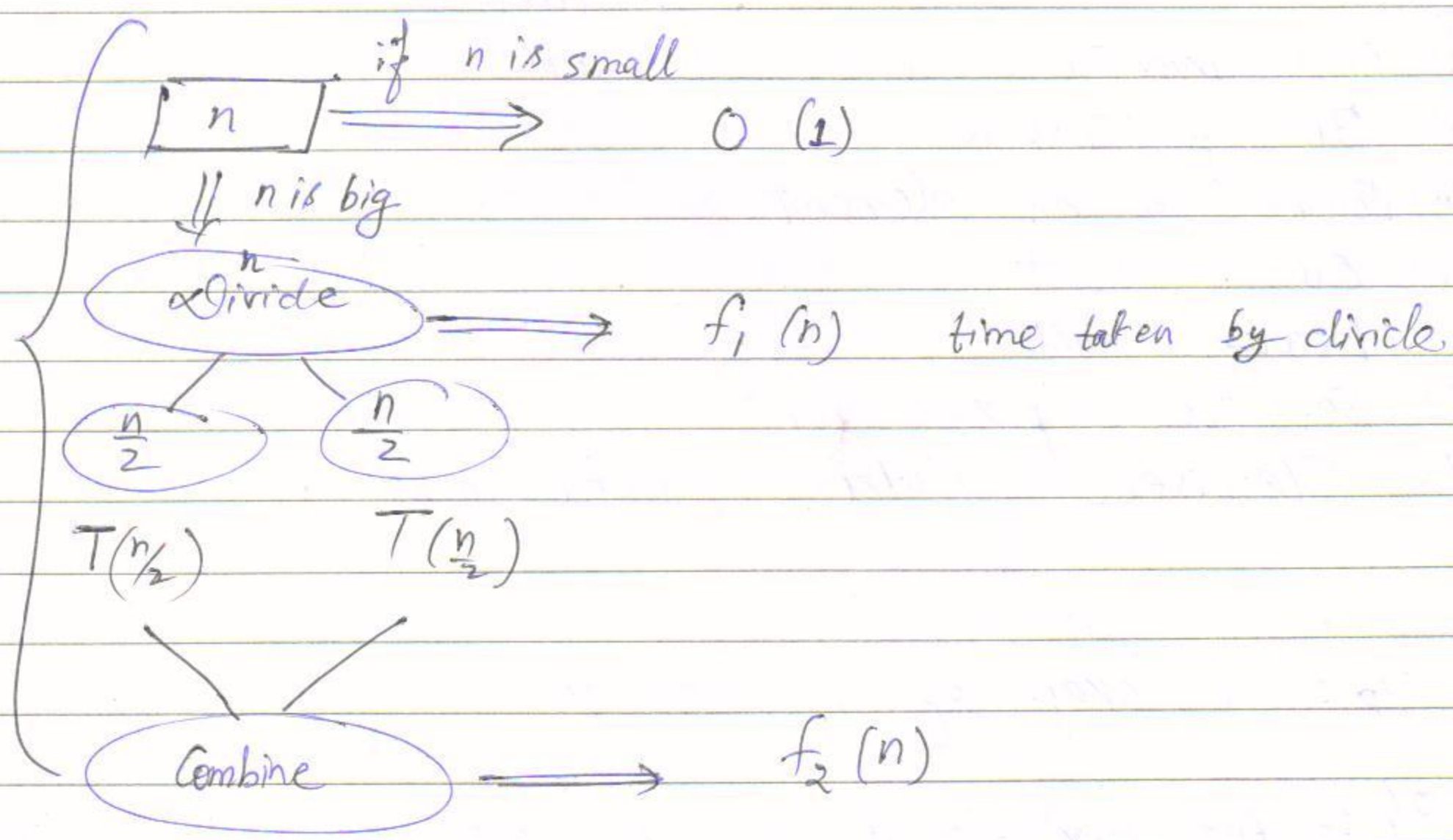
→ small function code varies problem to problem.
→ Divide function also varies " " " .
→ in one problem combine function may be addition, in another may be multiplication, hence combine function also changes according to ~~funct~~ problem.

Control Abstraction ✓ — but code of all 3 control function is hidden Date ____ not given so Control Abstraction

↓

control ~~abst~~ flow is ① small function
② divide function
③ combine function

Q. How to ~~solve~~ find Time complexity of a problem if it is solved using devide & conquer.
TC = ?



$$T(n) = \begin{cases} O(1) & \text{if } n \text{ is small} \\ \cancel{f_1(n)} & \\ f_1(n) + T(n/2) + T(n/2) + f_2(n) & \text{if } n \text{ is big.} \end{cases}$$

$$T(n) = 2T(n/2) + f_1(n) + f_2(n)$$

$$= 2T(n/2) + f(n)$$

No — time taken to do divide

$= a \quad T(n/b) + f(n)$

Date

↓	↓	↓	$T(n) = 2T(n/2) + n\log n$
no. of. subprogram	size of subprogram	Divide & ~~Conquer~~ combine time.	$T(n) = 2T(n/2) + \frac{n}{\log n}$
			$T(n) = 5\ T(n/2) + n$
			$T(n) = 8\ T(n/2) + n$
			$T(n) = T(n/2) + n$

§ Application of Divide and Conquer:-

(i) Find maximum and minimum.
(ii) Binary search
(iii) Power of an element
(iv) Quick sort
(v) Merge sort
(6) Selection procedure.
(7) Strassen's Matrix multiplication.

(1) Find Max Min :-

i/p : An array of n- elements.

o/p :- Find max element and min element.

if given array is sorted $O(1)$ { return array[0] → min; return array[n-1] → max.

if given array is not sorted :-

eg:- i/p : A[10, 20, 30, 1, 2, 3, 11 21 31]
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

o/p : max = 31
min = 1

Date

⇒ Straight max min (

~~min = 10~~

max = ~~10~~ ~~20~~ 30 ~~30~~ ~~30~~ 31

min = 10 10 ~~10~~ ~~1~~ 1

max = ~~10~~ ~~20~~ 30 31

min = ~~10~~ 1

By me:-

```
Straight max min ( A , n )
{
    int max, min;
        max = min = a[i] ;  // assume array starting index is
  for(i=2 ; i<=n ; i++)
    {
      if(a[i] > max)
        max = a[i] ;  break ; A
                      return B
                      exit C.
      else
      if (a[i] < min)
        min = a[i] ;
A-  }
  return (max, min) ;  ------> return through array
                               [min] max.
}
-B
```

```
main
{
}
C
```

Let T(n) be the no. of comparisions required to find max and min ~~to~~ of into given array of n element using straight max min. algo.

Best case → No failure, every time less work.

$T(n) = 1 \cdot (n-1)$

Worst case → every time failure {every time both conditions check every time max work

$T(n) = 2 \cdot (n-1)$

Date

average case :- uniform distribution

$$T(n) = 1.\left(\frac{n-1}{2}\right) + 2\left(\frac{n-1}{2}\right) = 3\frac{(n-1)}{2} = 1.5(n-1)$$

if 5 (if conditions) in the loop then

average case :-

$$T(n) = 1.\frac{(n-1)}{2} + 2.\frac{n-1}{2} + 3.\frac{(n-1)}{2} + 4.\frac{(n-1)}{2} + 5.\frac{(n-1)}{2}$$

P. Average $A(n)$
Worst $W(n)$
Best $B(n)$

T & F ?

(a) ✓ $A(n) = O(W(n))$ T
(b) $A(n) = o(W(n))$ ∝
(c) $W(n) = O(B(n))$ ✗
(d)

Time complexity :-

Best case

$T(n) = 1.(n-1) = n-1 = O(n)$

Worst case

$T(n) = 2.(n-1) \Rightarrow O(n)$

Space Complexity :-

if array is taken in non recursive program then only $O(n)$.

if 2D array $O(n^2)$

max = 2B
min = 2B
i = 2B
6B extra space } 3 local variable

5 Comp ⇒ so $O(1)$

[1 element
  2 element] in array ⇒ small problem

Q. Find Max and Min in given array of n elements. using recursision. or divide and conquer.

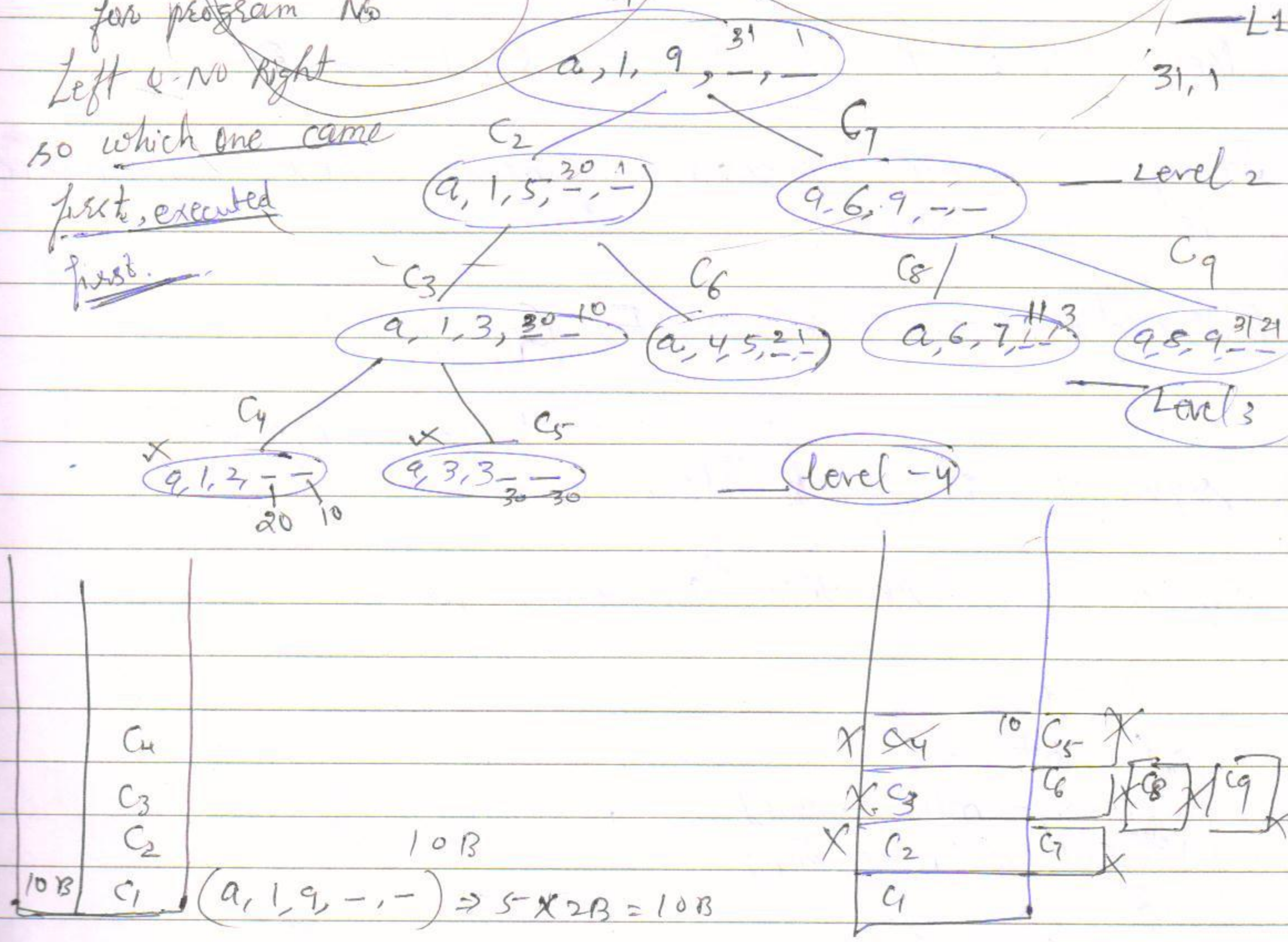
Date

Sol

i/p : A [10 (1) 20 (2) 30 (3) 1 (4) 2 (5) 3 (6) 11 (7) 21 (8) 31 (9)]

10,10 30,30 2,1

for program No Left & No Right so which one came first, executed first.



Space complexity of recursive program:-

No of levels in tree.

4 × 10 ⇒ $\log_2 n \times 10$ ⇒ $\log_2 9 \times 10$ ⇒ $10 \times \log n = O(\log n)$

Time complexity of recursive program

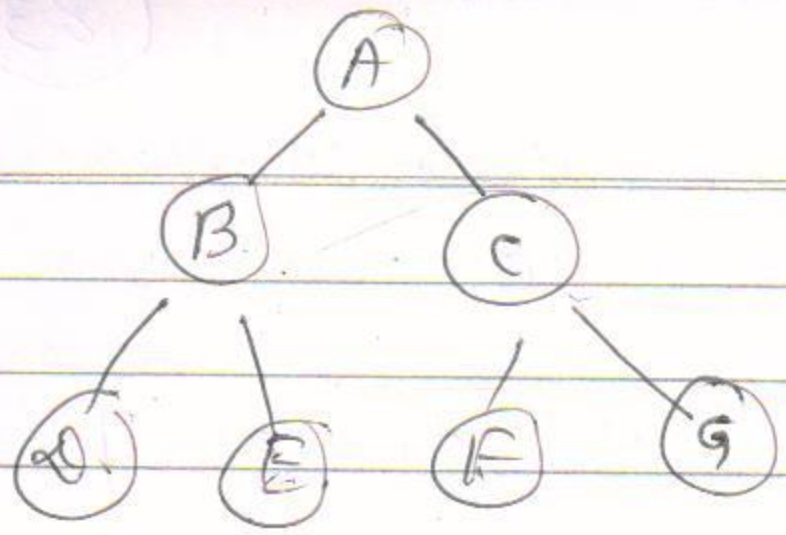
No. of functions called —

here C_9 = 9 functions.

for $n = 9$ = 9 function

Date

Q



Pre order : A, B, D, E, C, F, G,

In programming languages, function calling sequence is pre-order.

Post order :- D, E, B, F, G, C, A

function evaluation sequence is post order.

```
Divide & Conquer Max Min (a, i, j, max, min)
{
  if (i = j)
  {  int mid;
     max = min = a[i];
     return (max, min);
  }
  if ( i == j-1)
  {
     if (a[i] > a[j])
        max = a[i], min = a[j];
     else
        max = a[j], min = a[i];
     return (max, min);
  }
```

1-elem: no comparision

2 elem: 1- comparision

Small

Date

```
else
{
  mid = (i+j)/2 ;  } — Divide  0 - comparision
  OAC maxmin ( a, i, mid, max1, min1) ;   ⟶ T(n/2) comparision
  OAC max min (a, mid+1, j, max2, min2) ;  } Conquer
                                            ⟶ T(n/2) comparison
  if (max1 > max2)  ⟶ 1 - comparision
    max = max1 ;
  else
    max = max2 ;

  if (min1 < min2)  ⟶ 1 - comparision
    min = min1 ;
  else
    min = min2 ;
  setuen (max, min);
}
```

Big problem

combine. ⟶ 2 Con

Let T(n) be the total no. of comparisions required b/w the elements to find max and min in the given array of elements using Divide and conquer maxmin algo.

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ 1 & \text{if } n=2 \\ 0 + T(\frac{n}{2}) + T(\frac{n}{2}) + 2 & \text{if } n>2 \end{cases}$$

$$\Downarrow$$

$$T(n) = 2T(\frac{n}{2}) + 2$$

Using substitution method — get Time complexity

Date

$$T(n) = 2T(n/2) + 2$$

$$= 2\left[2T(n/2^2) + 2\right] + 2$$

$$= 2^2 T(n/2^2) + 2^2 + 2$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + 2^3\right] + 2^2 + 2$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2$$

$$= 2^k T\left(\frac{n}{2^k}\right) + 2^k + \cdots + 2^2 + 2$$

have to stop. TC. $\Rightarrow \frac{n}{2^k} = 2 \Rightarrow n = 2^{k+1}$

$\log n = k+1$

$k = \log n - 1$

reverse →

$$= 2^k T\left(\frac{n}{2^k}\right) + 2^1 + 2^2 + 2^3 + \cdots + 2^k$$

$$= 2^{\log n - 1} T\left(\frac{n}{2^{\log n - 1}}\right) + 2^1 + 2^2 + 2^3 + \cdots + 2^{\log n - 1}$$

G.P.

$$= \frac{2^{\log n}}{2} T(2) + \left\{ 2\,\frac{(2^{\log n - 1} - 1)}{2-1} \right\}$$

$$= \frac{n}{2} \cdot 1 + 2\left(\frac{n}{2} - 1\right)$$

$$= \frac{n}{2} + n - 2 \Rightarrow \frac{3n}{2} - 2 \Rightarrow 1.5n - 2$$

Order $O(n)$

$n = 2 \cdot 2^k$

$O(\log n)$ Stack space.

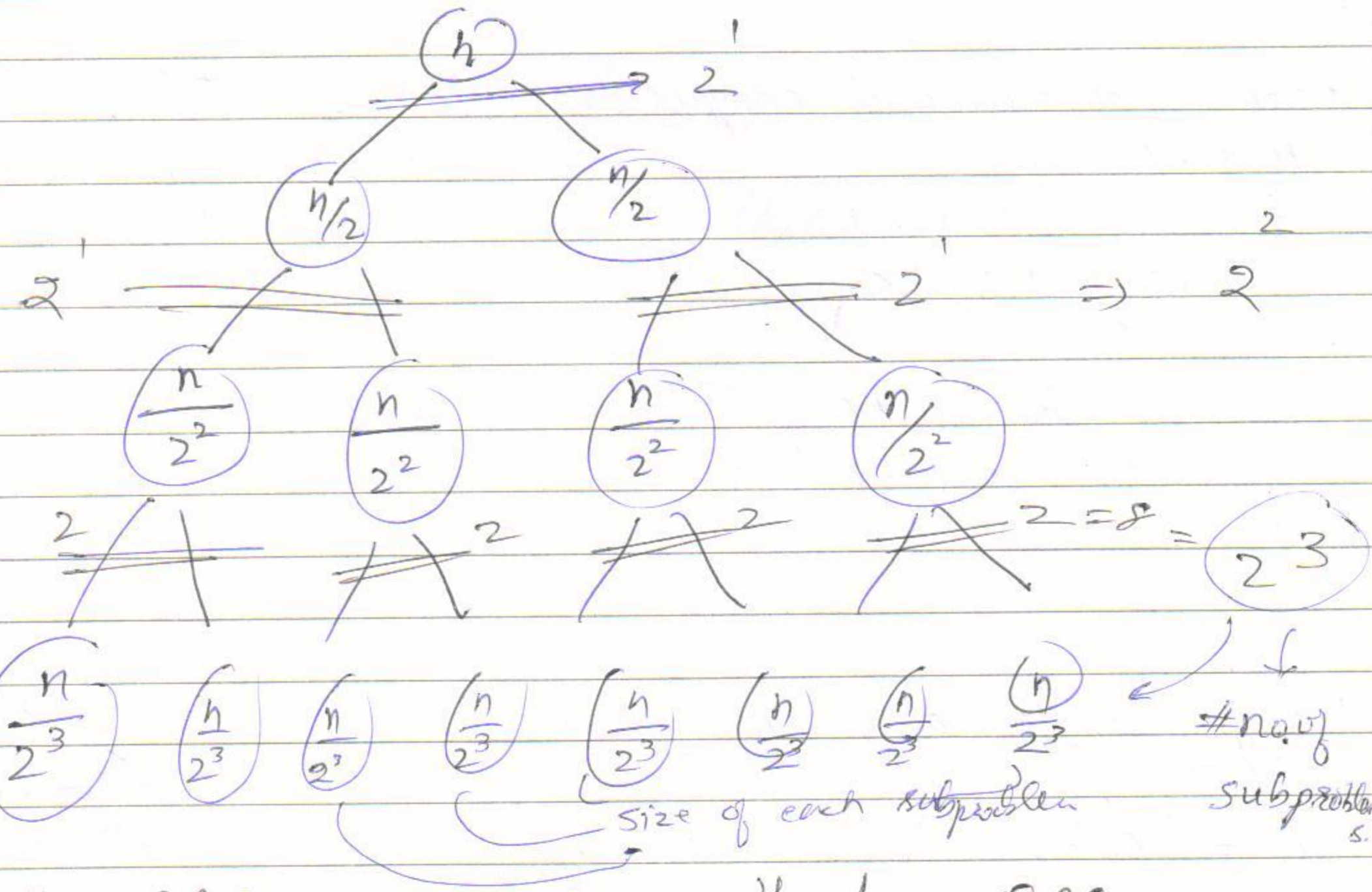
recurs.

Date

In average case it is same.
means — in Best case same
Worst case same
averag cas same.

so $\boxed{TC = O(n).}$

$T(n) = 2T(n/2) + 2$



with DAC

Time = $O(n)$
Space = $O(\log n)$

without DAC

Time : $O(n)$
Space : $O(1)$

Note:- Using div & C & without DAC Date
it is ~~using~~ taking same time so we have to consider space also, without using DAC, it is best bcoz it is taking $\theta(1)$ space.

§ Power of an element:-

i/p: 2 +ive integers a && n ($a \geq 1$ && $n \geq 0$)
o/p: return (a^n)

without divide and conquer:-

```
Power(a, n)
{   int f=1, i;
    for(i=1; i<=n; i++)
    {
    f = f * a;
    }  printf(f);
}
```

= n * 1 multiplication ✓
= n - Mul
= n * O(1) ⇒ O(n)

time complexity.

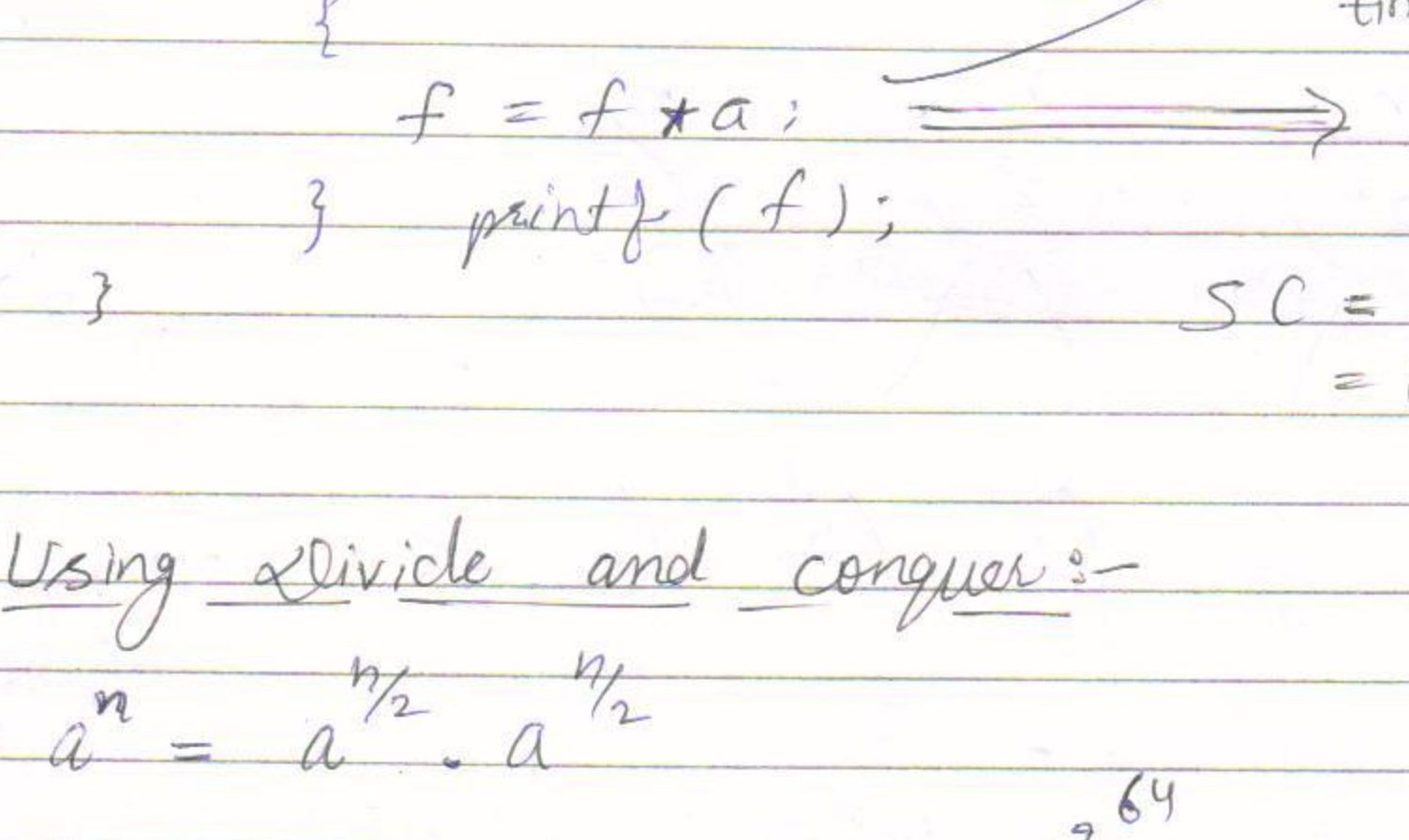
$\Longrightarrow \theta(n)$ [B.C, W.C, A.C]

SC = 4 B.
= O(1) space.

Using divide and conquer:-

$a^n = a^{n/2} \cdot a^{n/2}$

T



multiplication sequ
$2^{64} = 6$
$2^{16} = 4$
$2^{128} = 7$
$2^n = \log n$

Date

```
Power (a, n)
{
  if (n==1) return (a);
  else
    divide n = n/2
    return (power (a, n) * a)
}
```

```
Power (a, n)                                    0 M
{
  if (n==1)                                     }
    return (a);                                 }
  else                                          } divide
  {mid = n/2;                                   }        T(n/2)
  b = power (a, mid);                                    T(n/2)
  b = b * b;        ⇒ 1 multiplication
  return (b);
}
```

Let $T(n)$ be the no of multiplications required to find a^n using DAC then

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 0 + T(n/2) + 1 & \text{if } n > 1 \end{cases}$$

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n \geq 1 \end{cases}$$

Date

$$= T\left(\frac{n}{2}\right) + 1$$

$$= \left[T\left(\frac{n}{2^2}\right) + 1\right] + 1$$

$$= \frac{n}{2^3} + 1 + 1 + 1$$

$$= \frac{n}{2^3} + 3 \cdot 1$$

$$= \frac{n}{2^k} + k \cdot 1$$

$$= T\left(\frac{n}{2^k}\right) + k \cdot 1$$

$$= T(1) + \log n \cdot 1$$

$$= 0 + \log_2 n = \log n$$

$$= O(\log_2 n)$$

sum No series.

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log n = k \log 2$

Note:- in else part no, break, no exit, no return & no condition so in best case, worst case and average case complexity is same so

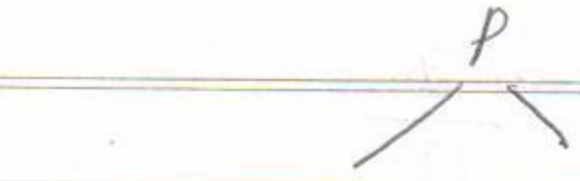
$$\boxed{TC = \theta(\log_2 n)}$$

2^1	
2^2	
2^4	
2^8	
2^{16}	
2^{32}	} a & b local so depratly come
2^{64}	} a, n, mid, b,

Space Complexity = $O(\log_2 n)$

p

Date



	A	B	
Time	n	log n	Time
Space → O(1)		log n	Space.

To find power of an element DAC algo is best algo bcoz it is taking log (n) multiplicati only.

~~§ Binary search :-~~

§ Searching :-

Linear Search :- No matter array is sorted or not.

i/p = An array of n-elements, element - x
o/p = return position of x if x is found
else
return (-1)

ex: i/p = 10 20 30 1 2 3 11 21 31
i 2 3 4 5 6 7 8 9

Op :-	x = 10	x = 3	x = 31	x = 50
	1	6	9	-1

false.

```
LS(a, n, x)
{
    int i;
    for( i=1 ; i<=n ; i++)
    {
        if(a[i] == x)
            return (i)
    }
```

Space: i only (2B) (local variable) O(1)

⇒ Best case : (No failure data)
T(LS(n)) = Ω(1)

⇒ Worst case :
T(LS(n)) = O(n)

⇒ Average case means

Date

average = equal probability $\frac{1}{n}$ ∴ n peopal are Case

$= \left(\frac{1}{n} * 1\right) + \left(\frac{1}{n} * 2\right) + \left(\frac{1}{n} * 3\right) + \left(\frac{1}{n} * 4\right) - - - + \left(\frac{1}{n} * n\right)$.

$= \frac{1}{n}\left[1 + 2 + 3 + 4 + - - - n\right]$

$= \frac{1}{n}\left\{\frac{n(n+1)}{2}\right\} = \frac{n+1}{2}$

= Average case of linear search $\frac{n+1}{2}$

another way:- for average case.

Total comparision = $1 + 2 + 3 + 4 - - - - + n$

$= \frac{n(n+1)}{2}$

$\text{average} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2}$

Space complexity = $O(1)$, ∵ 2B extra space

⇒ § Binary search :-

i/p : A storted array of n-elements, ele -x

o/p : Return position of x if x found
else
return (-1)

ex:- i/p : A[10 (1) 20 (2) 30 (3) 40 (4) 50 (5) 60 (6) 70 (7)]

o/p: x=40 | x=20 | x=10 | x=70

4

mid = $\frac{1+7}{2}$

BS(1,7)

$m = \frac{1+7}{2} = 4$

BS(1,3)

$m = \frac{1+3}{2} = 2$

Date

$x = 70$

B.S (1, 7)
4

BS (5, 7)
6

BS (7, 7)

7

```
B.S ( a , n , x )
{
   if ( n == 1)
   {
      if (a[i] == x)
         return (i);
   }
else
{
   mid = 1 + n/2 ;
   if (
```

```
BS (a, i, j , x)
 {    int mid;
  if (i == j)
   {
    if (a[i] == x)
       return (i);
    else
       return (-1);
   }
else
 {
   mid = (i + j)/2 ;
    if ( a[mid] == x)
       return (mid) ; }
```

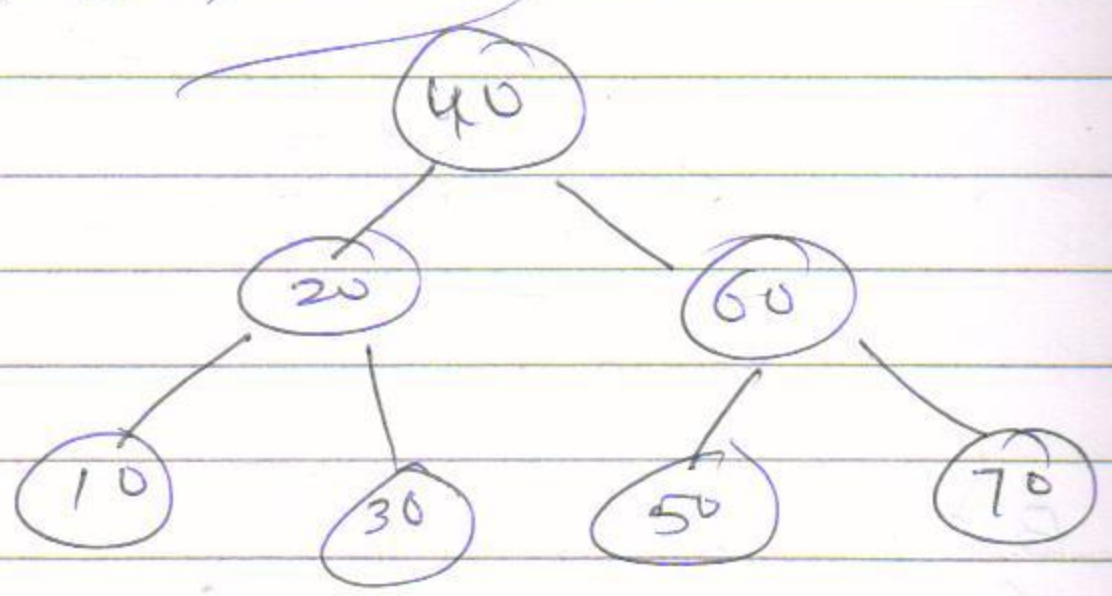
⇒ 1 comparision ⇒ O(1)

O(1)

O(1)

O(1) → 94 → T(n/2)

```
if (a[mid] < x)    BS(a, mid+1, j, x);
else
    BS(a, i, mid-1, x);  ⟹ T(n/2)
}
```



No of nodes = 7

$\log_2 7 = 2.81$

$= \underline{3}$ levels or comparisions.

⇒

Let T(n) be the time complexity of binary search algo on n elements then

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ \underbrace{O(1)+O(1)+O(1)}_{\text{Constant}} + T(n/2) & \text{if } n>1 \end{cases}$$

$T(n/2)$ → go only one side either left or right.

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ T(n/2) + C & \text{if } n>1 \end{cases}$$

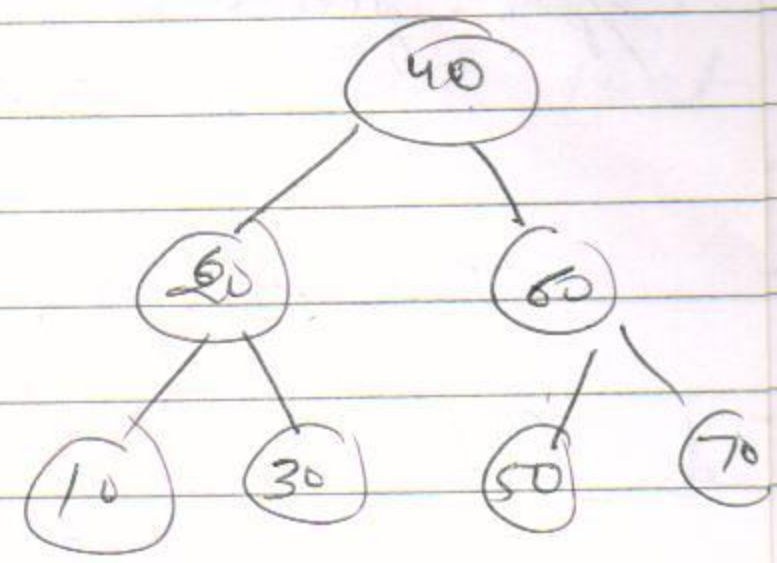
$= T\left(\frac{n}{2^2}\right) + 2C$

$= T\left(\frac{n}{2^3}\right) + 3C$

$= T\left(\frac{n}{2^k}\right) + k \cdot C$ ⇒ $\boxed{T(n) = O(\log n)}$ Worst case TC.

Worst
~~Best~~ case $TC = O(\log n)$

Best case $TC = 2(1)$



average case ⇒

$= 1 + 2 + 2 + 3 + 3 + 3 + 3$

total $= 1 + 4 + 12 = 17$

average $\frac{17}{7} = 2.4 = 3.$

So average case also $\boxed{TC = \theta(\log n)}$

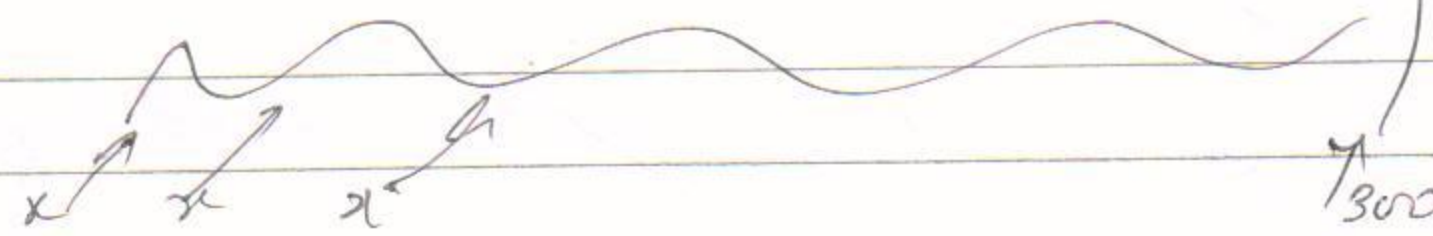
$TC = T(n/2) + C$

$T(n/2)$ → time to evaluate left or write

C → (time taken to work done to decide go whether Left or right.)

Among Linear search & Binary search.

Q. x is already present in an array of n element



(n − 1) comparision in worst case.

Q. i/p : a sorted array of n-elements. ✓ Date

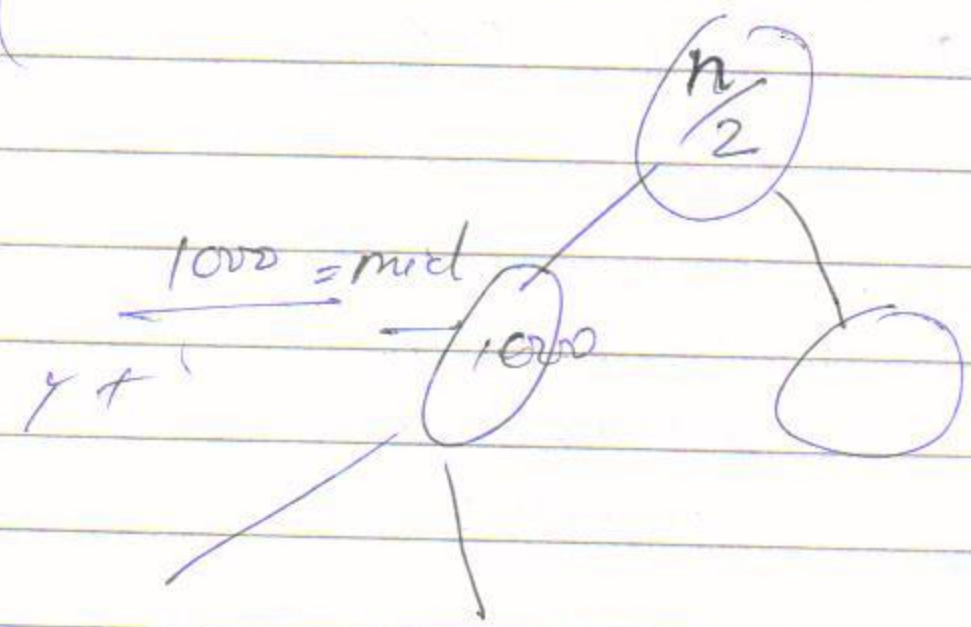
o/p : find any 2 elements a & b such that
$a + b > 1000$.

Best Algo (what is the worst case time comp.)

Ans Q(1)

Ans. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | . . . | n |

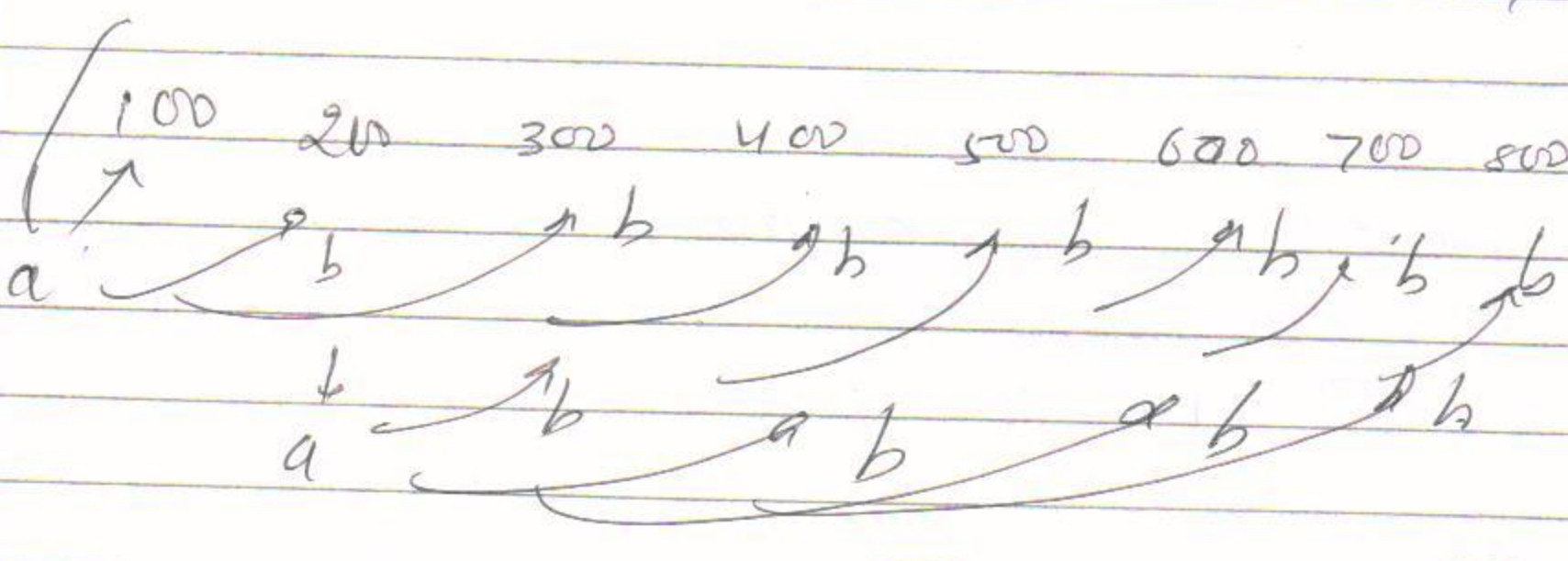
(100 200 300 400 500 600 700 800 900)



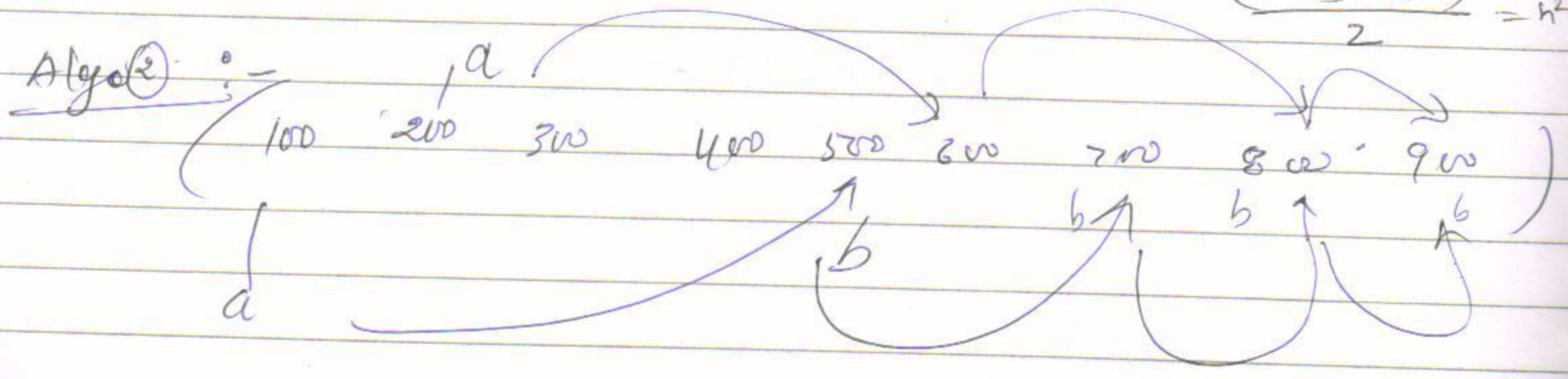
$\longrightarrow O(n^2)$

$\Rightarrow n$ $\quad$ $\Rightarrow n$

Algo (1) :- for every element a apply Linear search to find another element b such that
$a + b > 1000$



Algo (2) :-



(n log n)

Algo 2:- $\xrightarrow{n}$ Date $\log n$

For every element a apply binary search to find another element b such that $a+c > 100$

Algo 3:- return (a[n-1], a[n]),
else
return (-1).

$\Rightarrow \theta(1)$

[BC = W.C. = A.C.]

Algo -4:-

Q. If Array is not sorted

Ans (i) sort array first $\rightarrow$ $n \log n$
(ii) Apply algo 3 $+ \ O(1)$

$O(n \log n)$

Algo 5: If array not sorted.

Best

(i) apply ~~Bub~~ 2 passes of bubble sort $\rightarrow$ $2n$
(ii) return (a[n-1], a[n]) $\rightarrow + \ O(1)$

$2n = O(n)$

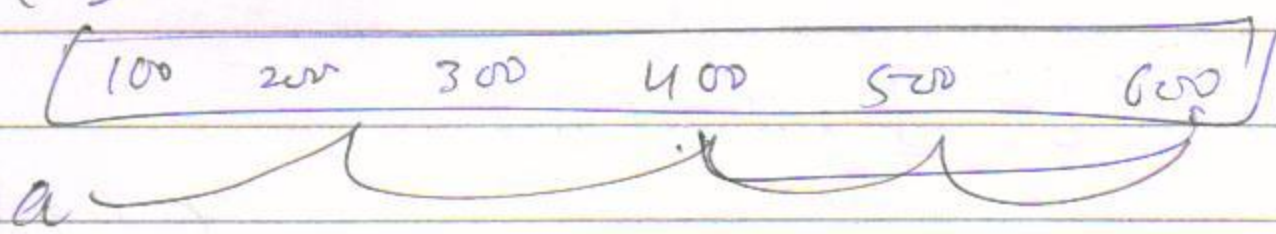
Q. If $a + b < 1000$ then

$n \log n > O(n)$

$n \times \log n \ @ \ O(n)$

Q. i/p : A sorted array of n elements Date_____
o/p: Find out any 2 elements a & b such that
$a + b == 10000$

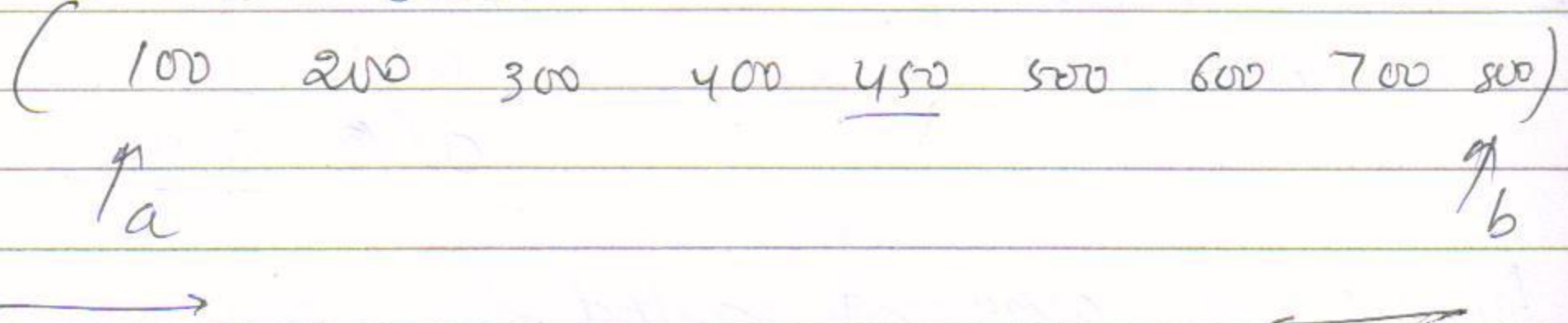
Ans O(n)



Algo (1) Using LS ⇒ $O(n^2)$

Algo (2) :- using Binary search ⇒ $O(n \log n)$

Algo (3) Greedy Algo :-



⇒ Algo {
① Take 2 variables A — 1'st of array
B — last of array
② If (A+B == C) return (A, B);
else
If (A + B > C) B = B − 1
else A = A + 1
It will take / continue max n times.

TC ⇒ O(n) → [for C. && A-C]

Sum of ~~subbeset~~ subset is 1000 where su

sum of subset problem ⟶ NP- complete problem

3 set problem.

which are distinct

Q. i/p: A sorted array of both +ve (or) -ve integers
o/p: find an ele a[i] such that a[i] = i

	-100	-75	-50	-25	-5	0	5	8	25	30	50	55	80 - 90	190
position →	1	2	3	4	5	6	7	8	9	10	11	12	13	14 15 16 17

Ans.

Algo (1) L.S: using L.S ⇒ O(n)

Algo (2) Binary search. ⇒ O(log n)

```
{
    position = mid = (i + j)/2;
    if (mid == a[mid])
        return (mid)
    else
        if (mid < a[mid])
            goto L.H.S.
        else
            goto R.H.S.
}
```

⇒ which are not distinct means

25 25 25
8 9 10

8 8 | 8 | 8 8 8
7

4 4
5

Gate 30-10-12

(100)

Q.

Date

i/p: An array of n-ele in which all elements are both +ive and -ve and untill some place and afterwards all are ∞.

o/p: Find the place of 1st ∞.

25	5	65	44	1	2	3	45	77	21	43	8	9	12	-5	-66	22	49	54	-64	-77	∞	∞	∞
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	...	37

Algo ① - L.S ⇒ O(n)

Algo ② ⇒ O(logn.)

Q. Now in above example if n value is not given - or unknown.

Algo - ① L.S = O(n)

Algo - ② Bin. S ⇒ O(logn).

n

1 2 3 4 5 6 9 . . . .

32

16

8

4

2

1

Divided

so Divide & Conquer
so Binary search applied.
TC = O(logn)

Q

Date

i/p: An array of n-ele all in the range from 1 to n
o/p: find repeated elements.

2	2	4	5	2	4	6	9	6	7	8	9
1											n = 10

Algo(1) :- TC = $O(n^2)$

Algo(2) :- (i) sort the array first ⟶ = $n \log n$

let array A

5	8	1	7	8	5	2	7

1	2	5	5	7	8	8

(ii) check adjusent element ⟶ = n

$n \log n + n$

$O(n \log n)$

Algo 3

① for (i = 1; i ≤ n; i++) } $O(n)$
b[i] = 0

A

5	8	1	7	8	5	2	7
1	2	3	4	5	6	7	8

B

1	0	0	0	1	0	0	1
1	2	3	4	5	6	7	8

② for (i = 1; i ≤ n; i++)
{
if (b[a[i]] == 0)
b[a[i]] = 1
else
pf(a[i]);
}

= $O(n)$

8, 5, 7

2n

TC = $O(n)$

Q.

Date

i/p: An array of n-element in which every element
0 (OR) 1

o/p: sort the array.

Soln:-

0	1	1	0	1	0	1	0	1	0
1	2	3	4						0

① Algo 1 :- Simplest existing ⇒ $n \log n$

Algo 2 :- ① int $c_0 = c_1 = 0$;
for ($i = 1$; $i \le n$; $i++$)
{
if ($a[i] == 0$)
c_0++;
else
c_1++;
}

② for($i = 1$; $i \le c_0$; $i++$) c_0
$a[i] = 0$;
for($i = c_0 + 1$; $i \le n$; $i++$) $c_0 + c_1$
$a[i] = 1$;

$O(n)$ + $O(n)$ = $O(2n)$ = $O(n)$

0	0	0	0	0	0	1	1	1	1	1
0					c_0	c_0+1				n

& Merge sort :-

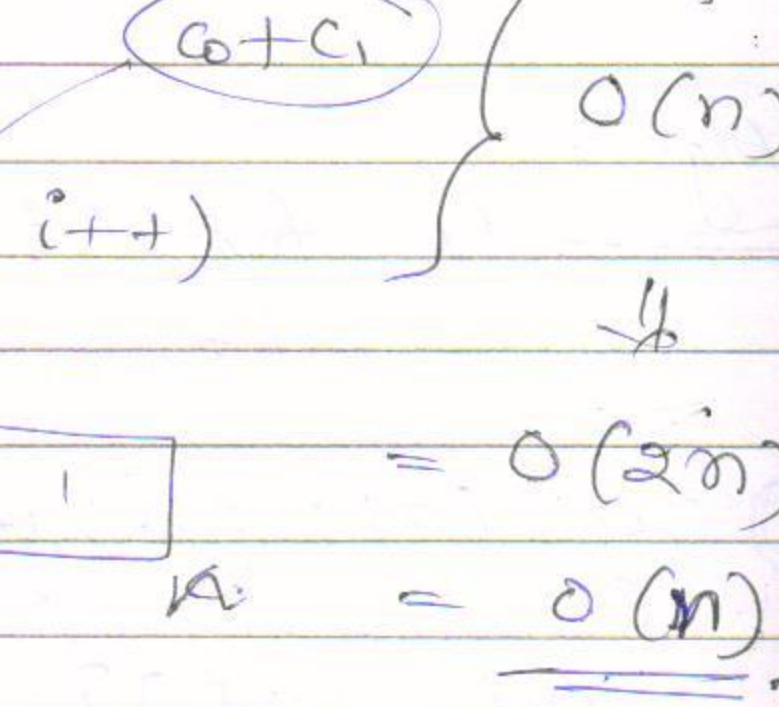
Note :- Merging 2 sorted subarrays.

← should be power of 2

Ex:-

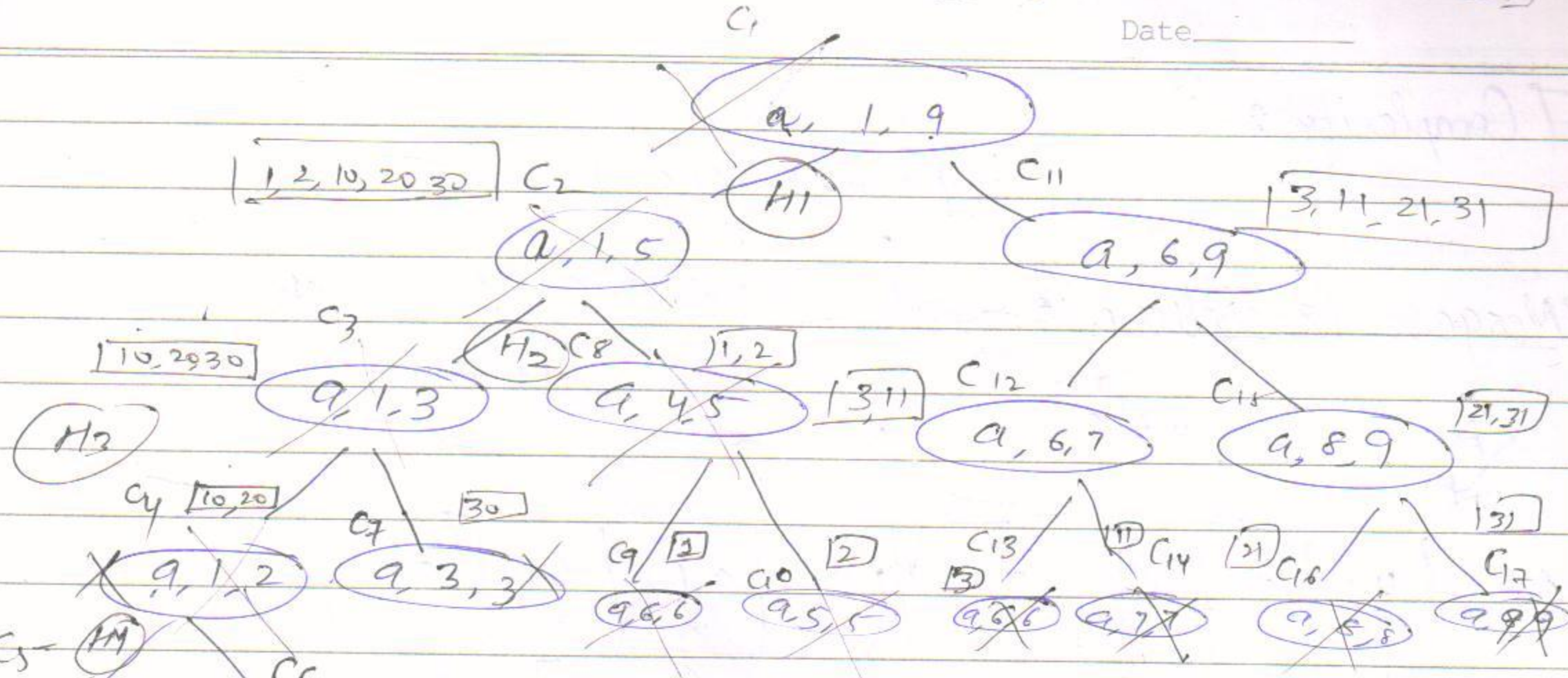
i/p: A

10	20	30	1	2	3	11	21	31
1	2	3	4		6	7	8	9



1, 2, 3, 10, 11, 20, 21, 30, 31

Date

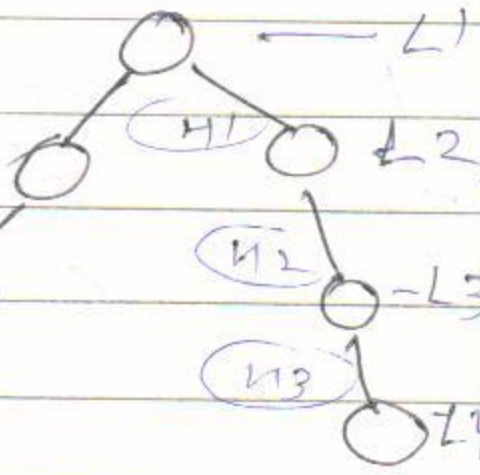


C5	C6
C4	C7 C9 C10 C13 C14 C16 C17
C3	C8 C12 C15
C2	C11
C1	

Hight of tree :- The longest distence from root node to leaf node.

OR level - 1

eg → 3



In above example -

H3 is Hight ⇒ L level - 1

4 - 1 = 3

Call by reference :- on the behalf of a group or array just send one parameter or base address of array.

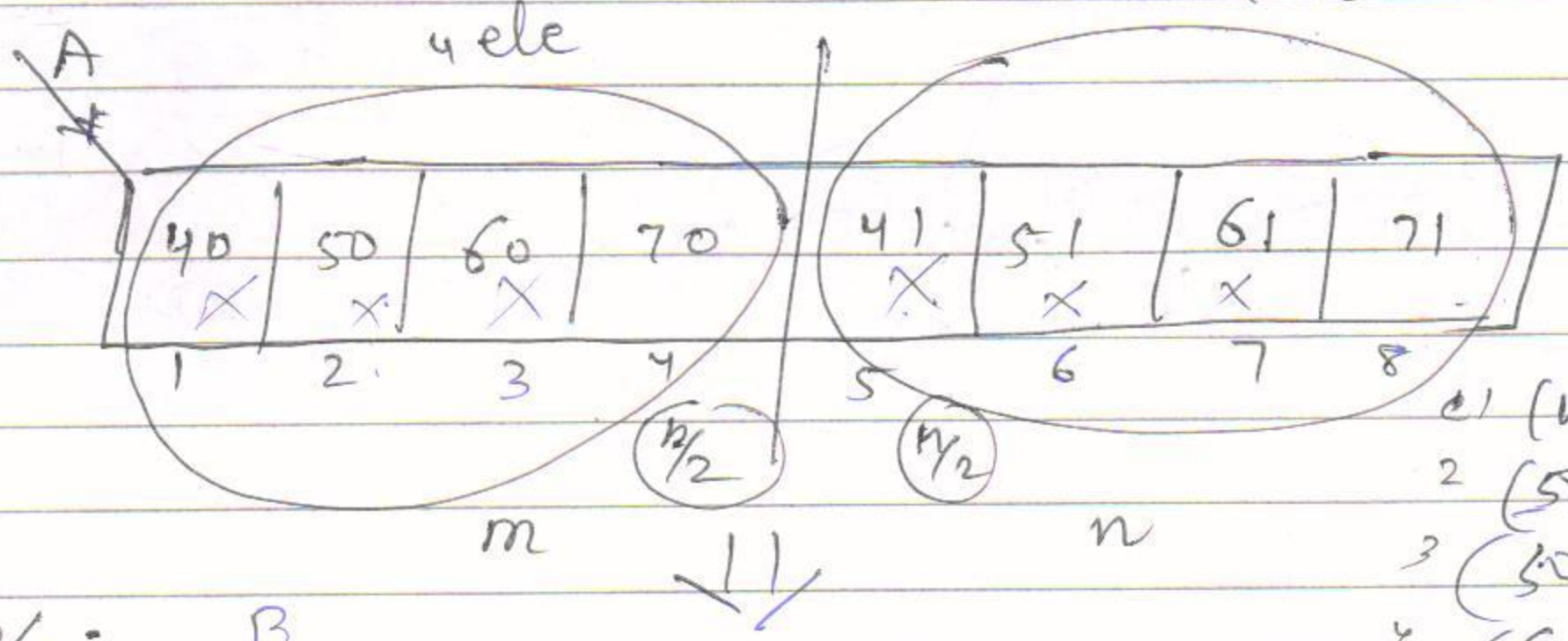
Call by value :- When only few variables or entity in group send all the val

Date

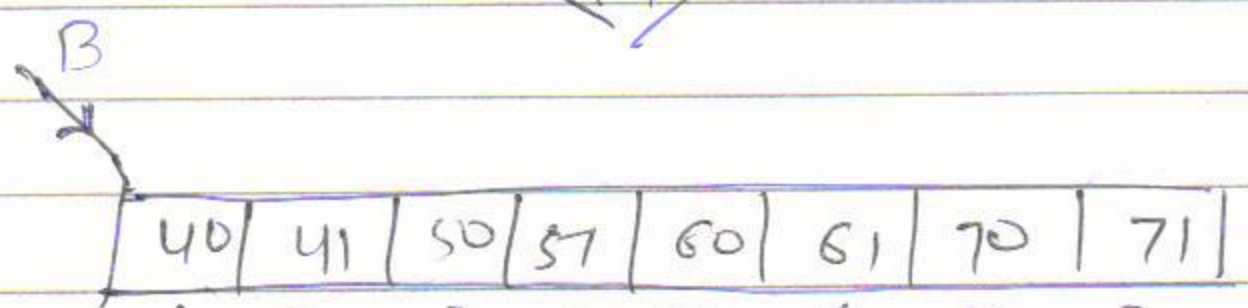
T Complexity :-

→ Merge Procedure :- [worst case]

i/p : Two sorted subarrays.



o/p :



1 (40, 41) = 40
2 (50, 41) = 41
3 (50, 41) = 50
4 (60, 51) = 51
5 (60, 61) = 60
6 (70, 61) = 61
7 (70, 71) = 70

$4 + 4 - 1 = 7$

$n/2 + n/2 - 1 = n - 1$

$m + n - 1 = m + n - 1$ → Total comparision

In worst case no. of comparisions ⇒ $(m+n-1)$ OR $(n-1)$

⇒ No of element movement for 2 subarrays into on sorted array

$4 + 4 = 8$

$n/2 + n/2 \Rightarrow n$

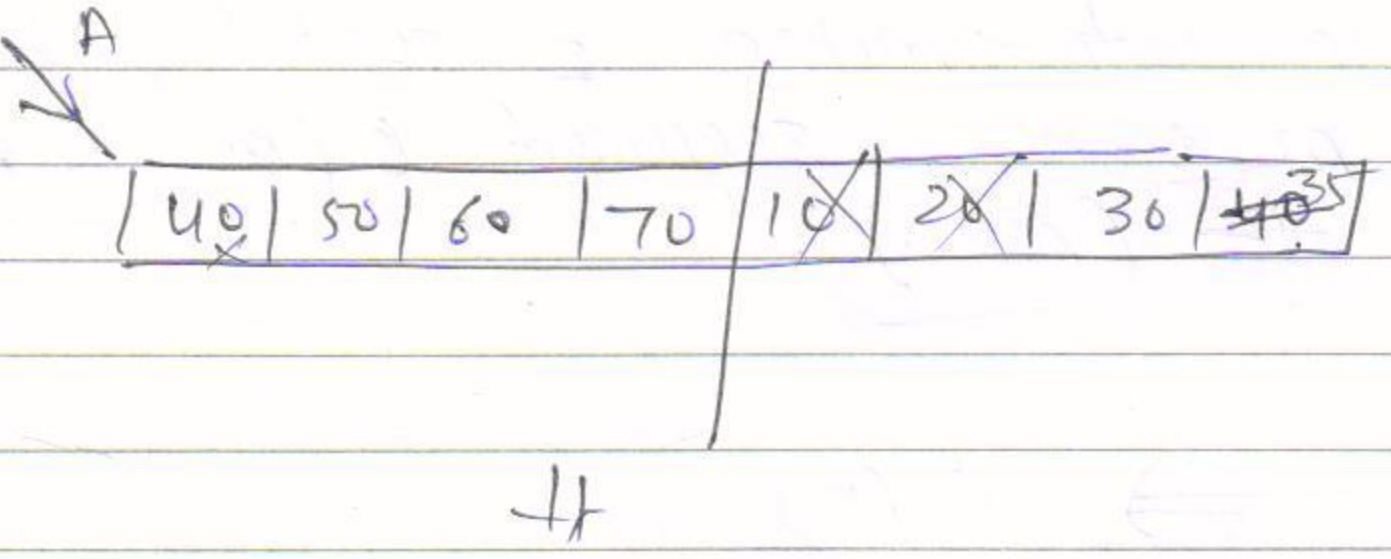
$(m + n)$ all are moved.

so time complexity ⇒

Time Complexity = $O(m+n)$

Date

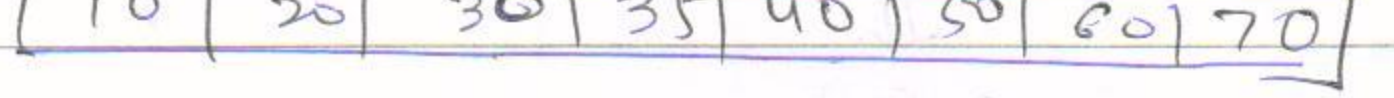
⇒ Merge - procedure [Best Case]:-



only 4 comparision

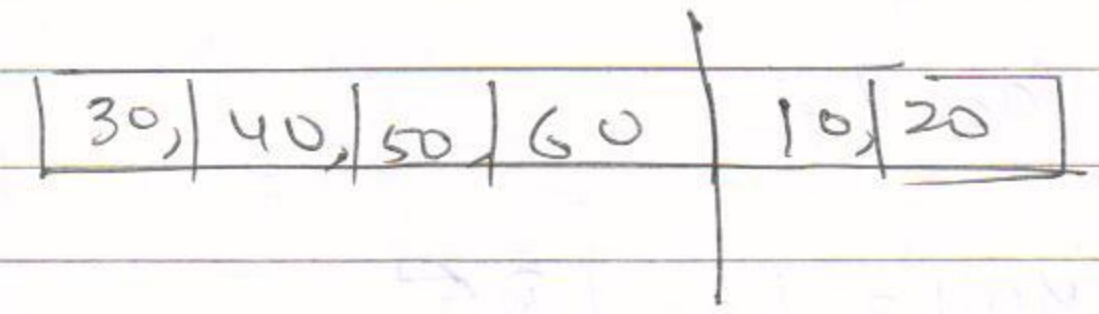
(40, 10) = 10
(40, 20) = 20
(40, 30) = 30
(40, 35) = 35

40
50
60
70
} No Comparision

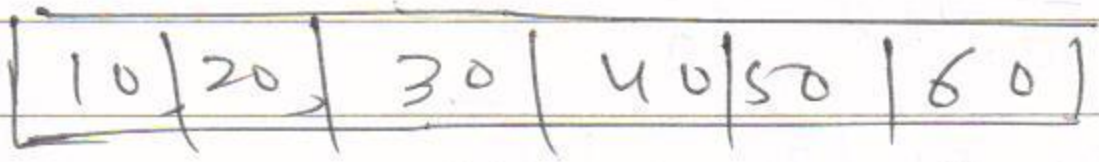


1st group + 2nd group ⇒ 4 Comparision only.
4 4

n/2 n/2
m n ⇒ [minimum (m, n)]



(30, 10) = 10
(30, 20) = 20



No of Comparisions ⇒ minimum (m, n)
here min (4, 2)

Comparison (2)

Movements :-

4 + 4
n/2 + n/2
m + n ⇒
movemens

so
4 + 4
n/2 + n/2
m + n ⇒ min (m, n)

⇒ O(m+n)

in exames generally asked comparision ⇒ but Time complexity is No of movement.

NOTE: In order to merge 2 sorted subarrays of size m & n, required $O(m+n)$ time,

∴ ($\underline{O(m+n)}$, $\underline{2(m+n)}$.

└ time Complexity of movements

$\frac{n}{2}$ $\frac{n}{2}$ ⟹ $O(n)$
$2(n)$
↓
$O(n)$

} movements ↓

Space :- Extra space ⇒ an array of ~~n~~ n elements
so extra space ⇒ $O(n)$

⇒ Out place sorting technique :- If any sorting Algo take an extra array (or) memory to sort. the array elements

⇒ In place sorting technique :- If No extra space is required

Merge - procedure [In place] :-

n/2 n/2

| 40 | 50 | 60 | 70 | 40 | 20 | 30 | 35 |

40

| 10 | 50 | 60 | 70 | 20 | 30 | 35 | 40 |

| 10 | 20 | 30 | 35 | 40 | 50 | 60 | 70 |

10 —— n/2
20 —— n/2
30 —— n/2
35 —— n/2

n/2 time ⇒ $\frac{n}{2} \times n/2$

⇒ $\frac{n^2}{4}$

= $\frac{1}{4} n^2$

Time Comm ⇒ $O(n^2)$

Date

⇒ Merge [In place] ⇒ $\frac{n}{2}, \frac{n}{2}$ ⇒ $\frac{n^2}{4}$

time Complexity ⇒ $O(n^2)$

Space Complexity ⇒ No extra space

so [O(1)]

⇒ Merge [out place] :—

① time complexity ⇒ $\frac{n}{2}$ $\frac{n}{2}$ ⇒ $\theta(n)$ T.C.

(movements)

② Space Complexity ⇒ $\theta(n)$

⇒ Merge sort(a, i, j)

```
{   int mid;
    if (i==j)                    → O(1)
      return (a[i]);
    else
    {                            divide
      {mid = (i+j)/2;            → O(1)
      {Merge sort (a, i, mid);   → T(n/2)
      {Merge sort (a, mid+1, j); → T(n/2)
      {Merge (a, i, mid, mid+1, j); ⇒ θ(n)
       return (a);
    }
}
```

Conquer

combine

B

[| | | | |]

& again copy in array a

a

[| | | | |]

Let T(n) be the time complexity of merge sort algorithm on n elements then →

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + \theta(n) + O(1) & \text{otherwise if } n > 1 \end{cases}$$

Date

$$T(n) = 2T(n/2) + n$$

$$= 2\left[2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n$$

$$= 2\left[2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + \frac{n}{2}\right] + n$$

$$= 2\left[2^2 T\left(\frac{n}{2^3}\right) + \frac{n}{2} + \frac{n}{2}\right] + n$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + n + n + n$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

$$\Rightarrow 2^k T\left(\frac{n}{2^k}\right) + kn$$

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log n = k$

$$\Rightarrow 2^{\log n} T(1) + \log n \cdot n$$

$$\Rightarrow n^{\log n} + n \log n$$

$$= n + n\log n$$

$T(n) = O(n \log n)$ = This is outplace

Now Inplace Merging

$$T(n) = O(1) + 2T(n/2) + O(n^2)$$

$$T(n) = 2T(n/2) + n^2$$

Date

$T(n) = 2T(n/2) + n^2$

$= 2[2T(n/2^2) + (\frac{n}{2})^2] + n^2$

$= 2^2 T(\frac{n}{2^2}) + \frac{n^2}{2^1} + \frac{n^2}{2^0}$

$= 2^3 T(\frac{n}{2^3}) + \frac{n^2}{2^2} + \frac{n^2}{2^1} + \frac{n^2}{2^0}$

$\downarrow$

$= 2^k T(n/2^k) + n^2 \left[\frac{1}{2^2} + \frac{1}{2^2} + \frac{1}{2^0}\right]$

$= 2^{\log n} T(1) + n^2 [O(1)]$

$= n(1) + n^2$

$T(n) = O(n^2)$

In case of In-place Mergesort
Space complexity
$= O(\log n)$

so finaly $TC = O(n^2)$ In-place Mergesort.

and space = extra space = n + $\log n$
(extra array in case of out) (stack space)

out spce $\longrightarrow$ $= \theta(n)$

NOTE 1:-

⇒ Merge sort is not in place because it is taking $o(n)$ extra space in the merge procedure.

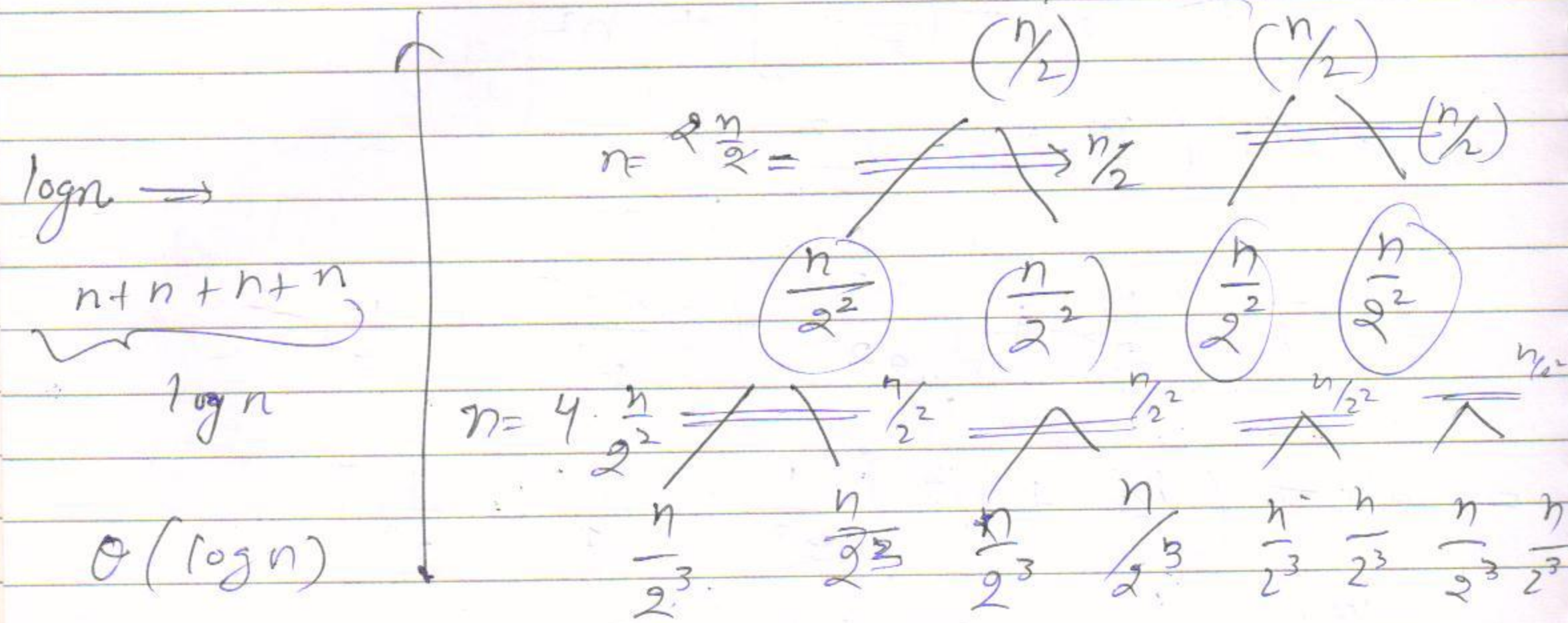
NOTE 2:-

Merge sort also is (not) advisable for smaller size array

[Insertion sort is advisable for smaller size array]

$T(n) = 2T(n/2) + \Theta(n)$

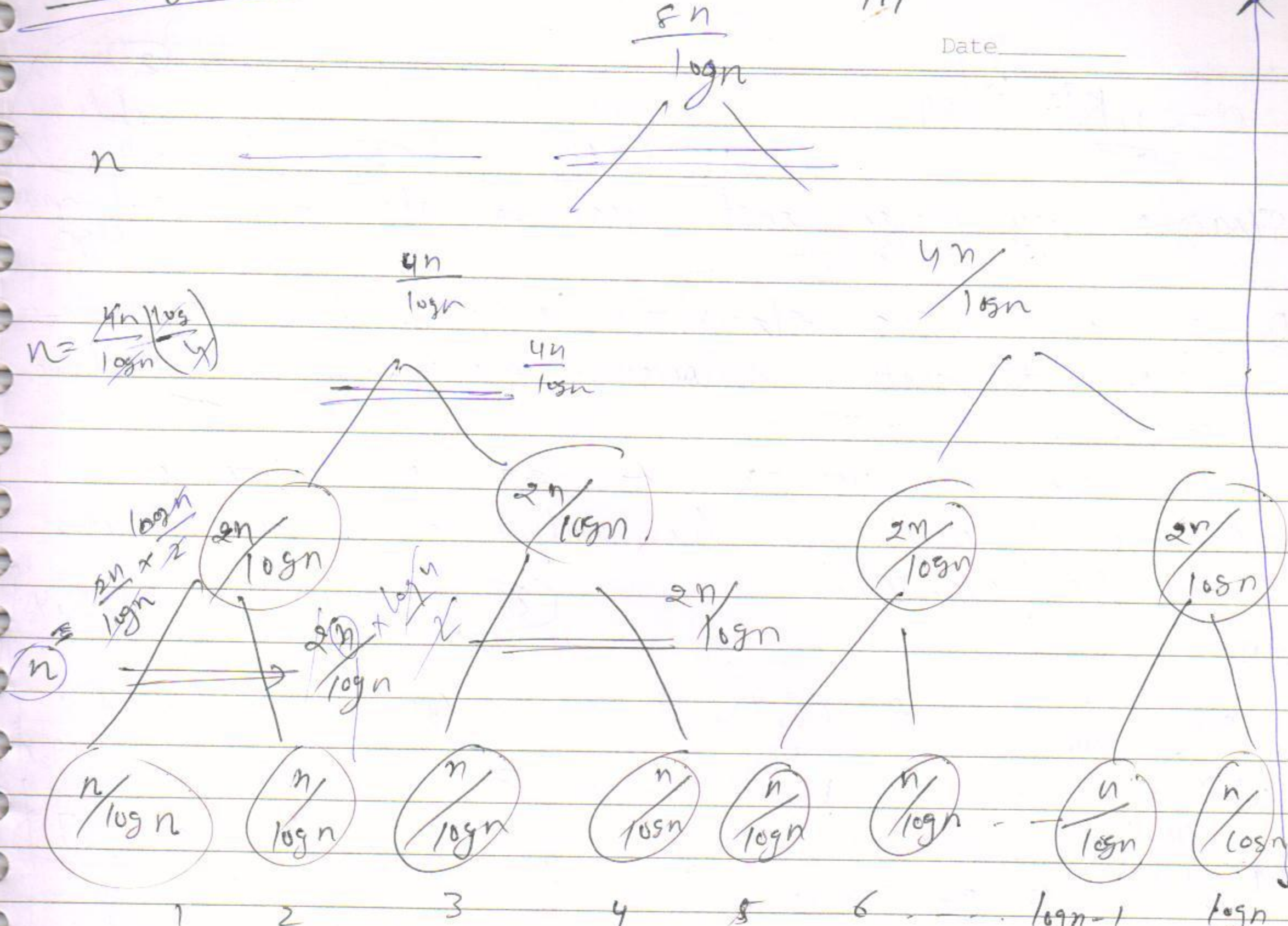
Date


n
(n) combine time
n/2
n/2
n/2
n/2
n = 2 n/2 =
n/2
n/2
n/2
n/2

$n \times \log(\log n)$

111

Date

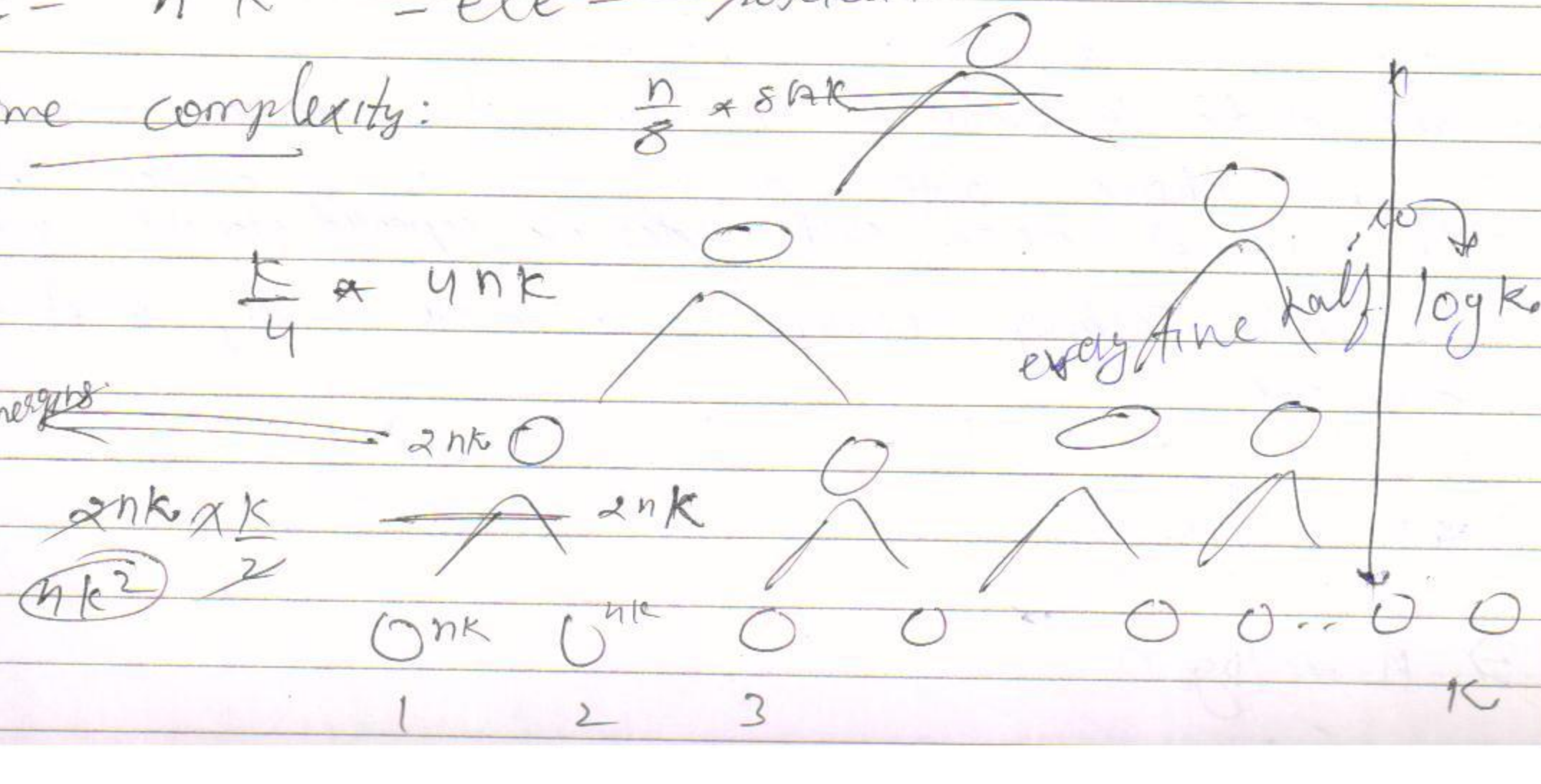


Ex.2 sorted

i/p ⇒ k – sub array each size – nk.

o/p:- nk^2 – ele – sorted.

Time complexity: $\frac{n}{8} * 8nk$


E/4 * 4nk
k/2 merges
2nk
2nk * k/2
4k²
2nk
2nk
2nk

$= nK^2 (\log K)$ → every time Half 1/2

if there is pass 1 & pass 2 means not $O(nc)$ problem that is simple Algo

Straight 2-way Merge sort means ⇒ take 2 element and merge

Q. If one uses straight 2-way Merge sort algo to sort following ele — in ascending order.

20 15 45 35 5 27 16 10 7 5

then what will be the output after second pass of the algorithm.

Ans. (10)

(5a, 5b 7, 10, 16, 20, 27, 35, 45)

8 ⇒ 2 ⇒ (5b 7)

o/p of 3rd pass. (5a 10, 16, 20, 27, 35, 45)

(10) ⇐ ⇒ 8

o/p after 2nd pass. (15 20 35 45) (5a, 10, 16, 27)

(10) ⇐ ⇒ 4 ⇒ 4

o/p after 1st pass (15 20) (35 45) (5a 27) (10 16) (5b 7)

(10) ⇐ ⇒ 2 ⇒ 2 ⇒ 2 ⇒ 2 ⇒ 2

20 15 45 35 5 27 16 10 7 5

merge sort is a stable sorting technigonique.

→ stable sorting technique :- if order of repeated element not changes after sorting then it is called S.S.T.

↳ In case of merge sort order of repeated element is not change

⇒ No stable sorting Technique :- if order of repeated element changed.

⇒ $10 * \log_2 10$

⇒ $n * \log n$

⇒ $O(n \log n)$

Date

Q. Find common

A	10	20	30	40	50	60	$\Rightarrow n$
B	5	15	20	25	30	35	$\Rightarrow n$

intersection

$A \cap B$: 20, 30 [using merge algo]

Time: $\Rightarrow O(n+n)$

$\Rightarrow O(2n)$

$\Rightarrow O(n)$

Q. Find Union (mean all)

10	20	30	40	50	60
5	15	~~20~~	25	30	35

small prime

$A \cup B$: 5 10 15 20 25, 30, ~~40~~, ~~50~~ 35, 40 50, 60.

$O(n)$

2. Quick sort [Tony Hoare] :-

Date

(1) Most practicaly used sorting algo
2.) one of the apllication of DAC.
3) In place [No requires extra space]
4) Not stable [means order of repeated element changes in this algo]

★ without temp

20
30 10
a=10, b=20

a = a+b
b = a-b
a = a-b

④ Less space
④ More cpu time

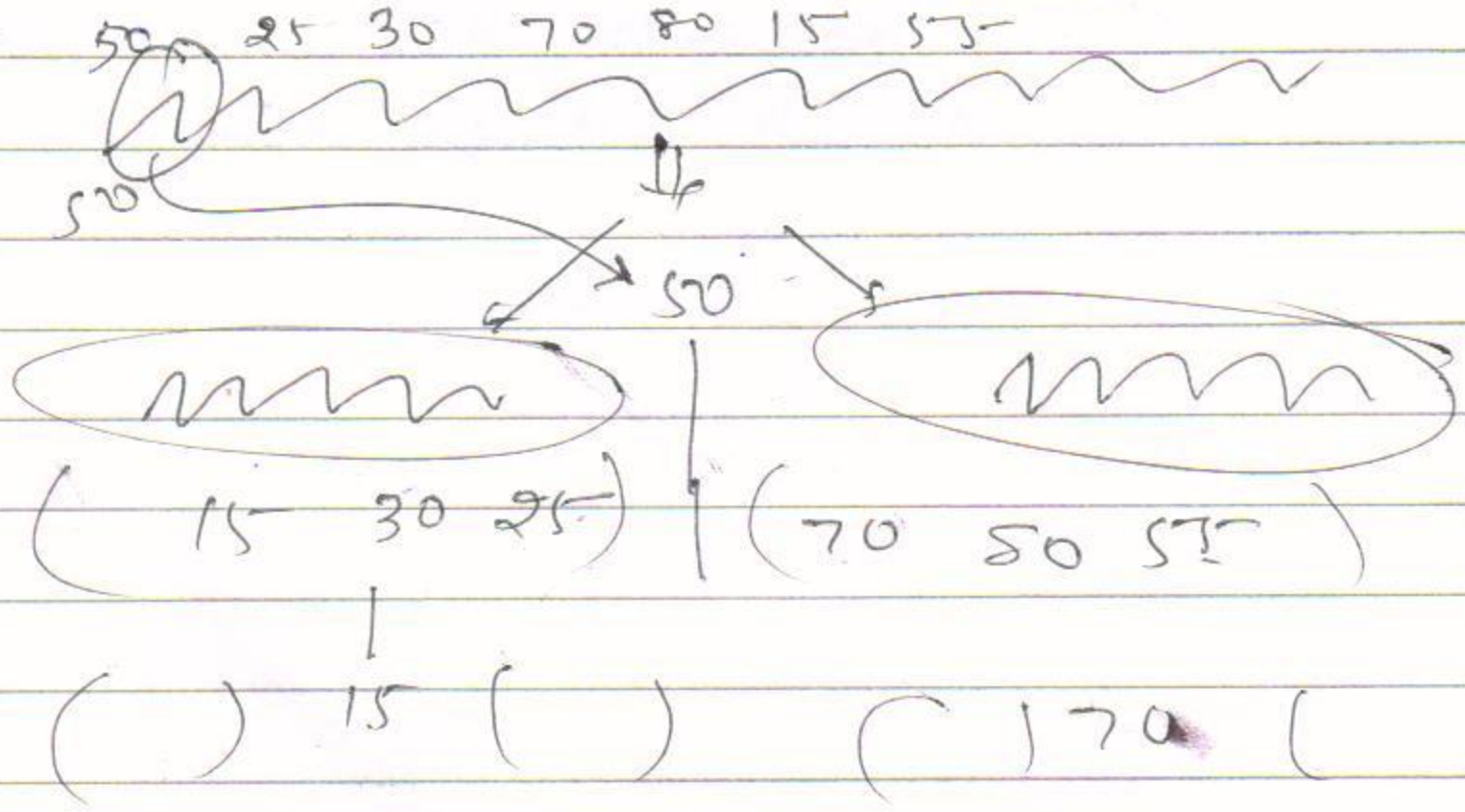
with Temp

a=10 , b=20
t=a
a=b
b=a

★ More Space
★ Less CPU time

Quick - sort algo
↓
Partition algo.

Partition Algo :- concept



Date

Partition :- or i → 1st element of array, or j → last element of array

```
Partition (a, p, q)
{
    x = a[p];        // choose first element for partition.
    i = p;           // called pivot element.
    for(j = p+1; j ≤ q; j++)
    {
        if (a[j] ≤ x)
        {
            i = i + 1;
            swap (a[i], a[j]);
        }
    }
    swap (a[p], a[i]);
    return (i);
}
```

for random generation of pivot value:
(x = RG(p, q)
  x = a[r]);

Comparisions = always (n-1) = O(n)

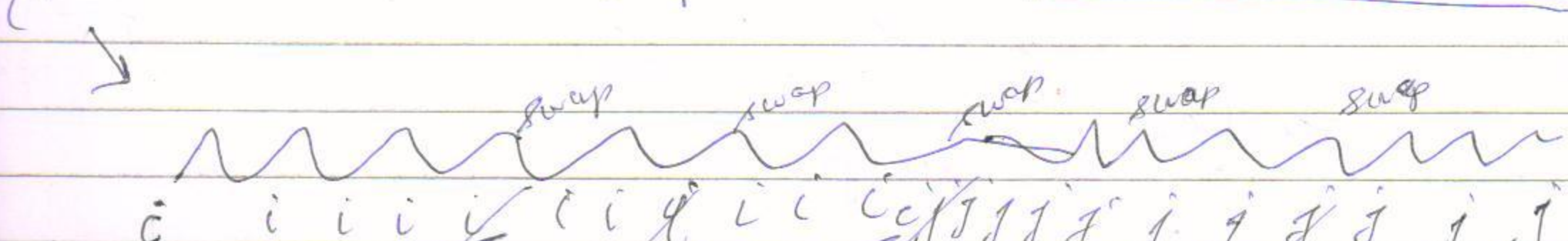
swap ⇒ n[W.C]
1[B.C]

In place.

⇒ **Randomised Quick Sort Algo :-**
if every time we choose random pivot element so that will be called randomised quick sort algo.

⇒ **Simple Quick sort Algo :-** Pivot point is always first element of subarray or any fixed element.

eg ⇒ i (Less than 50)
j (More than 50)



Step 1): For loop

Date

10 22 17 5

1 25 15 50 75 77 80 75 77 80

a → 60 80 25 75 77 50 25 15 5 10 22 99 17

1 2 3 4 5 6 7 8 9 10 11 12

P → i j j j j j j j j j j q

Place | i j i i i i i i

smaller bigger

Step 2): swap

P → 60 | 1 25 15 5 10 22 17 | 75 77 99 80 ← q

P i

pivot | Smaller | bigger

Step 3): - return

// swap i & p

8 = (17 1 25 15 5 10 22) (60) (75 77 99 80)

1 2 3 4 5 6 7 8 9 10 11 12

i/p: 25 88 42 15 65 5 22 94

1 2 3 4 5 6 7 8

P → i j j j j j j j

Step 1 :- for loop :-

i j i

25 | 15 5 22 | 65 42 88 94

P | smaller | i | greater | j

swap

= (22 15 5) (25) (65 42 88 94)

1 2 3 4 5 6 7 8

P i

Note:- Date

We have not used extra array or space means in given array. only we have done sorting so in place sorting technique, is applied in Quick sort.

Time complexity :-

for loop running ⇒ (n-1) times.

so TC = O (n-1)

= O (n)

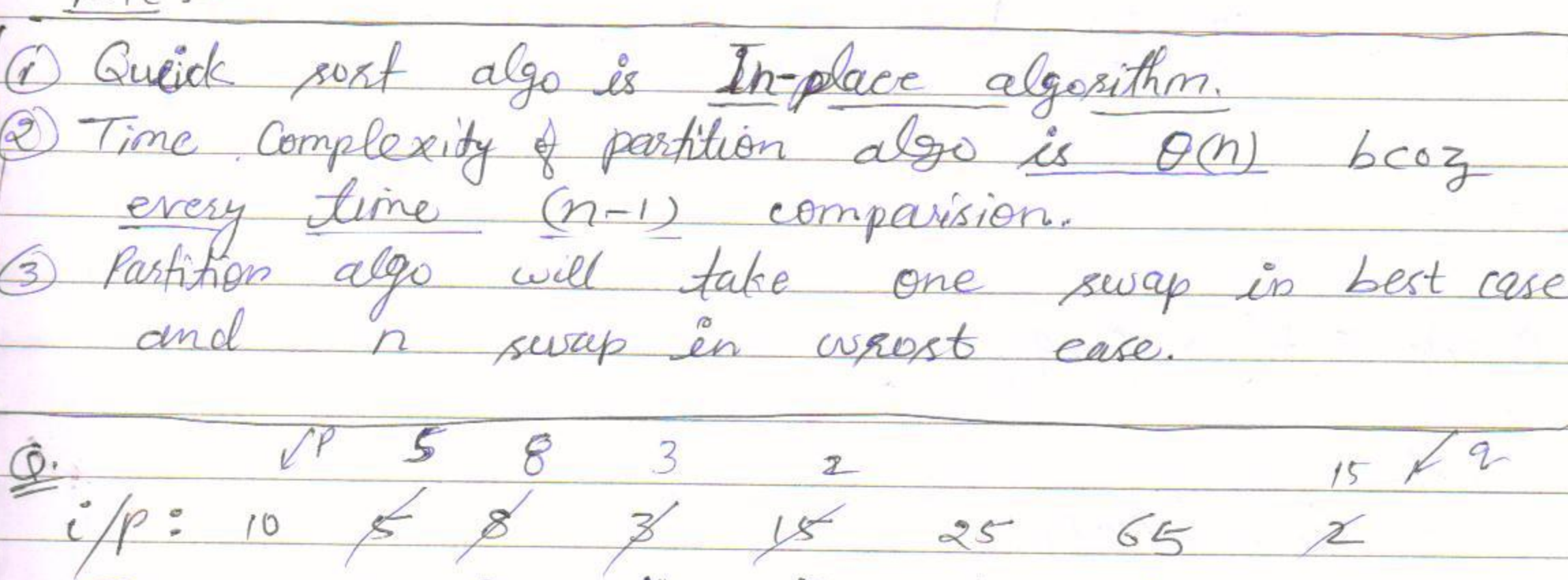
= -2 (n)

so ⇒ θ (n)

Note :-

(1) Quick sort algo is In-place algorithm.
(2) Time Complexity of partition algo is θ(n) bcoz every time (n-1) comparision.
(3) Partition algo will take one swap in best case and n swap in worst case.

Q. i/p : 10 5 8 3 2 15 25 65 2



if Pivot is fixed then Quick sort

118

Date

	10	20	30	40	70	50	60
P:	40	70	20	50	30	10	60
	1	2	3	4	5	6	7
		i	i	j	j	j	j
	i	i	i	(i)			

small → | greater

m

$m_4 = (10\ 20\ 30)\ (40)\ (70\ 50\ 60)$

1 2 3 | 4 | 5 6 7

Qs (1-3) | Qs (5-7)

(10) (20 30) | 40 | (50 60) (70) ( )

Qs 1 | 2 3 | 4 | Qs

Qs

$m_7 = (10\ (20)\ 30)\ 40\ (50\ (60)\ 70)$

1 2 3 | 4 | 5 6 7

⇒ Quick Sort :-

```
Quick Sort( a, p, q )        // p = 1, q = 100
{
    if ( p == q )
        return (a[p]);        } ⇒ O(1)
    else
    {
        m = partition ( a, p, q );
        QuickSort ( a, p, m-1 );
        QuickSort ( a, m+1, q );
        return (a);
    }
}
```

⇒ O(n)

one case	another case
$T(n/2)$	$T(m-p)$
$T(n/2)$	$T(q-(m-p+1))$

should be q

$= (m-p) + (1) + ((q-(m-p+1)))$

$= q$

Date

Let T(n) be the time compleaity of quick sort on n elements then

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \\ \theta(n) + T(m-p) + T(q-(m-p+1)) & \text{if } n > 1 \end{cases}$$

p ..... q

But

40 70 20 50 30 10 60 } May be

$\Downarrow$

$y = (20\ 30\ 10)\ 40\ (70\ 50\ 60)$ } May be

balanced

$\Rightarrow$

70 20 60 50 40 30 10 } May be

$\Downarrow$

$z = (20\ 60\ 50\ 40\ 30\ 10)\ 70\ (\ )$ } May be

10 70 20 30 60 50 40 } May be

$1 = (\ )\ \underset{1}{10}\ (\underset{2}{70}\ \underset{3}{20}\ \underset{4}{30}\ \underset{5}{60}\ \underset{6}{50}\ \underset{7}{40})$

Now all cases: $p\ \underset{\Downarrow}{(1-7)}$

0	1	6
1	1	5
2	1	4
3	1	3
4	1	2
5	1	1

} all combinations of partition.

divide (half half sort) or merge sort ← 3 1 3

Quick sort.

(Lucky L L L L - - - (R0)^{L})

Best Case :- if partition algo divide the array every time in exactly 2 parts $n/2$, $n/2$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ \Theta(n) + T(n/2 - 1) + T(n/2) & \text{due to pivot point} \end{cases}$$

$$= \Theta(n) + T\left(\frac{n}{2} - 1\right) + T(n/2)$$

So $T(n) = 2T\left(\frac{n}{2}\right) + n$

$T(n) = \Omega(n \log n)$ best case Time complexity

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2n$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

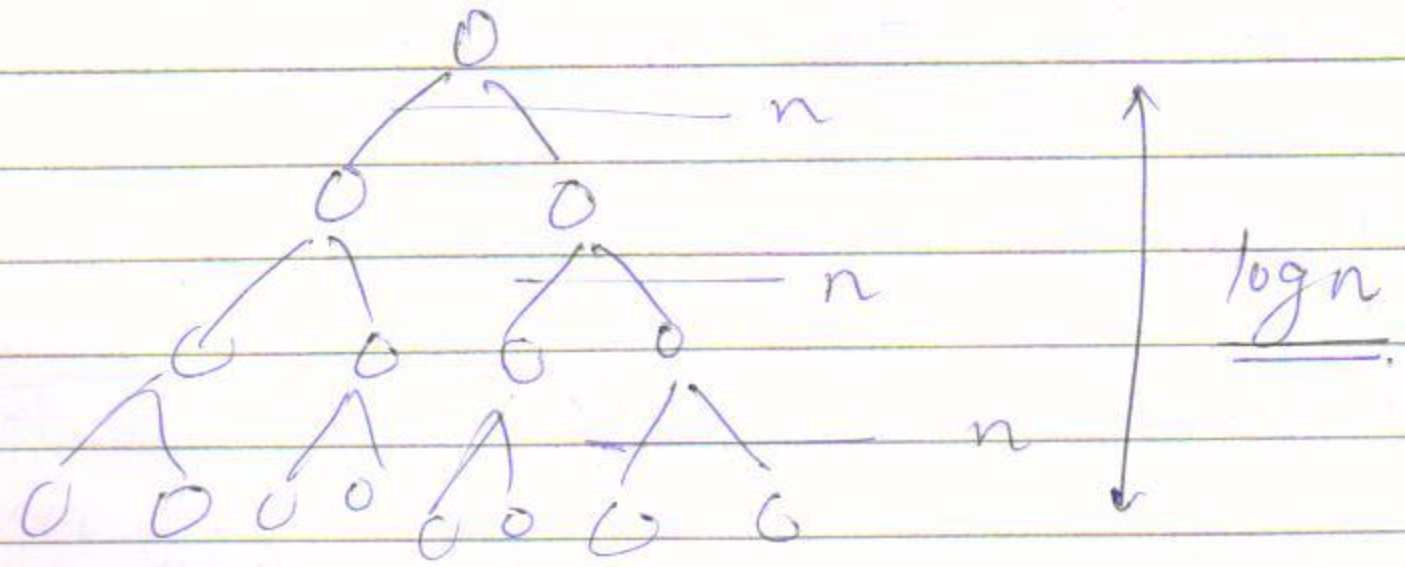
$\downarrow k$ $\quad \log n = k$

$T(n) = 2^{\log n} T(1) + \log n \cdot n$

$= n + n \log n$

$T(n) = \Omega(n \log n)$

Now Space Complexity :- space com = $O(\log n)$ // extra space is only stack blocks -



12) ( ) (Pivot) [70 35 60 90 71 74 35]
1 2 3 4 5 6 7 8

if in LHS 0 element & RHS remaining elements

Date

Worst Case :-

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ O(n) + T(0) + T(n-1) & \text{if } n>1 \end{cases}$$

$T(n) = T(n-1) + n$

$= T(n-2) + (n-1) + n$

$= T(n-3) + (n-2) + (n-0)$

$= T(n-k) + (n-(k-1)) + (n-(k-2)) \cdots + (n-0)$

$n-k = 1 \Rightarrow k = (n-1)$

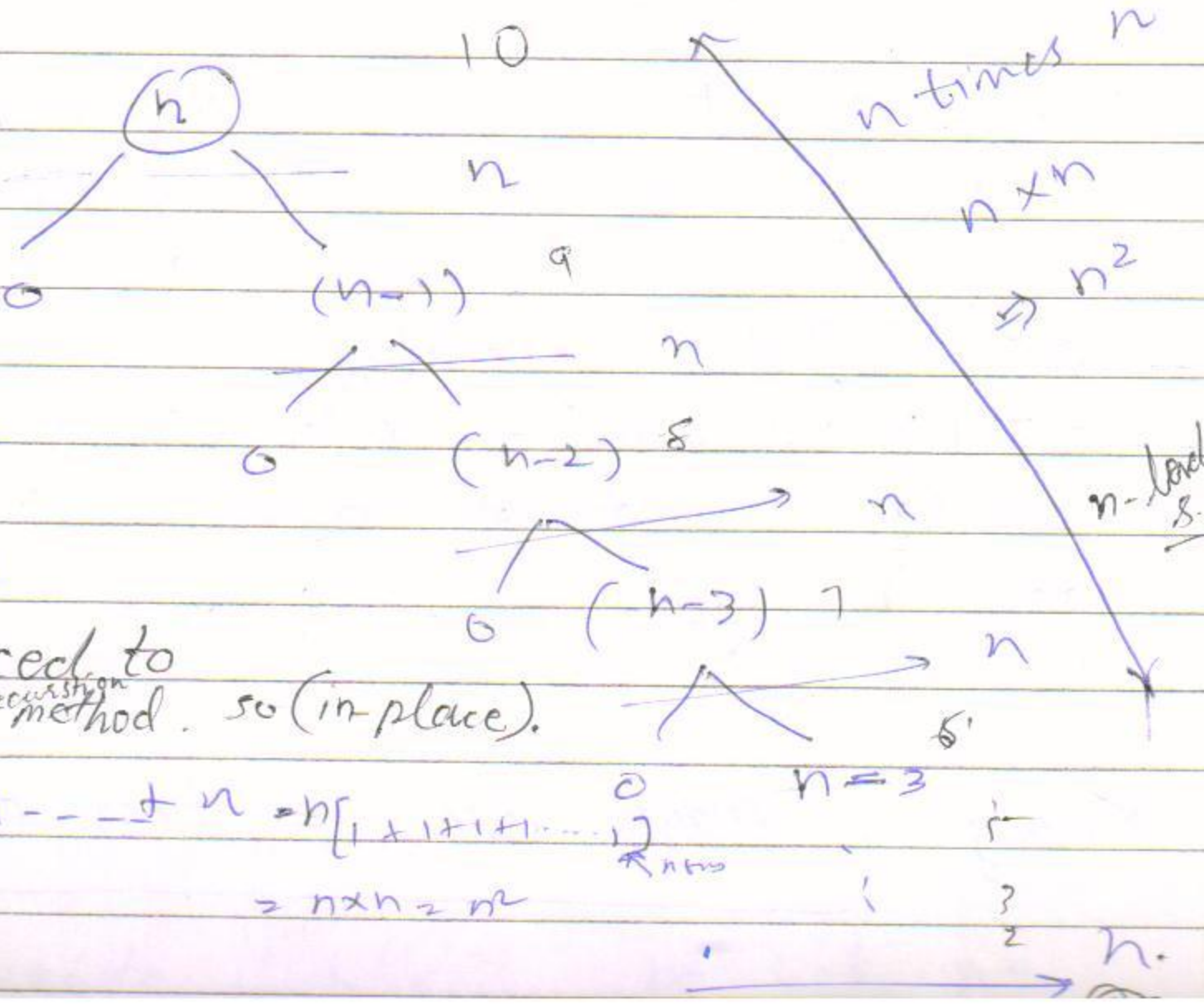
$= T(n-(n-1))$

$= T(1) + (n-(n-2)) + (n-(n-3)) \cdots + (n-0)$

$= 1 + 2 + 3 + 4 + \cdots + n$

$= n(n+1)/2$

$T(n) = O(n^2)$



here n levels are there so

Space Complexity =

extra space = O(n)

But this O(n) is reduced to O(log n) by tail recursion method. so (in-place).

$\Rightarrow n + n + n + n + n \cdots + n = n[1+1+1+1 \cdots 1]$ (n times)

$= n \times n = n^2$

$\Rightarrow O(n^2) \Rightarrow TC.$

So finaly: ({ every time if 0 && remaining element) such partition every time. U U U U U . . . U.)

122

Date

in Worst case Quick sort:-

$TC = O(n^2)$

$SP = O(\log n)$

⇒ Average Case :- (LU LU LU (OR) LU ..... LU = Lucky)

(UL UL UL UL . . . UL = Lucky)

Best

$T(n) = 2T(n/2) + n$

Wrost $T(n) = T(n-1) + n$

now average case → one time best, next time wrost

$$T(n) = 2T(n/2) + n$$

$$= 2\left[T\left(\frac{n}{2} - 1\right) + n/2\right] + n$$

$$= 2T\left(n/2 - 1\right) + 2\frac{n}{2} + n$$

$$= 2T\left(\frac{n}{2} - 1\right) + 2n$$

(constant, 1 com)

$$= 2T\left(\frac{n}{2}\right) + n$$

$T(n) = O(n \log n)$ ⇒ one best & one wrost finaly gives best case.

$= 2T(n/2) + n \Rightarrow n\log n$

$= 2T(n/2) + 2n \Rightarrow n\log n$

$= 2T\left(\frac{n}{2} - 1\right) + n \Rightarrow n\log n$

$= 2T(n/2) + 2n - 1 \Rightarrow n\log n$

Finaly | Average Case = Best case in Quick sort

Date

$T(n) = T(n-1) + n$ — 1st unlucky

$= 2T\left(\frac{(n-1)}{2}\right) + (n-1) + n$

$= 2T\left(\frac{n}{2}\right) + 2n - 1$

$= 2T(n/2) + n$

$= O(n \log n)$ — best again

Symbols:

$T(n) = \Omega(n \log n)$ } only symbol matters. in Quick sort
$T(n) = O(n^2)$ } then symbols → means best case
→ worst case.

now

Worst case of Quick sort is $\Omega(n \log n)$.

↳ when given in words — then — symbole doesn't matters

Avarge case of Q.S. is $\Omega(n \log n)$
$O(n \log n)$
$\Theta(n \log n)$

↳ symbol doesn't matter

Ex.

Q. Quick sort algorithm is run on 2 inputs shown bellow — To sort in ascending order.

(i) 1, 2, 3, 4, 5, ... n-1, n

(ii) n, n-1, n-2, n-3, ..., 2, 1

Let C_1 and C_2 be the no of comparisions made for the inputs (i) & (ii) respectively then which on of the following is true. ?

(a) $C_1 == C_2$

(b) $C_1 < C_2$

(c) $C_1 > C_2$

(d) Not - comparable.

	Comp	Swaps
Any input (i) ⇒	n^2	$n-1$
input (ii) ⇒	n^2	n^2

Unbalanced so worst case. 124

Date 9

Ans. p 1

10 | 20 30 40 50 60 70 ~~8~~ ~~9~~

i | j j j j j j ~~j~~ ~~j~~

x=1

↓ Partion on 7 elements ⇒ 6 comparison

( ) | (1) (20 30 40 50 60 70 ~~8~~ ~~9~~)

I=i | ↓ on 6 ele = 5 com

( ) | (2) (30 40 50 6 70 ~~8~~ ~~9~~)

| 5 = 4 com

i | 4 = 3 com

(n-1)) swaps. 3 = 2 com

2 com

1 com

$6+5+4+3+2+1 = \frac{n(n+1)}{2} = O(n^2)$

$n + \dots + 2 + 1 =$

$T(7) = 6 + T(6)$ ↓

$5 + T(5)$

$4 + T(4)$ ↓

$3\ T(3)$ ↓

$2.T(2)$

$1 + T(1)$ ↓

0

$$T(n) = \begin{cases} \\ n-1 + T(n-1) \end{cases}$$

$T(n) = T(n-1) + n$ comparision

$T(n) = O(n^2)$ → worst case.

Date

For (ii) input: - 5

P → 70 60 60 50 40 30 20 10 ← q

1 2 3 4 5 6 7

x=70 i j i j

i i i

10 60 50 40 30 20 70

70 60 50 40 30 20 10

j j j j j j

i i i i i

Worst Case swaps = (n)

comparisons = (n-1) (6)

10 60 50 40 30 20 70 (empty)

1 2 3 4 5 6

i j j j j j

j j j j

10 60 50 40 30 20

i j j

swap = (1)

comparisons (5)

20 50 40 30 (60) ( ) swap (n) (4)

( ) (20) (50 40 30)

30 40 (50) ( ) comparisions (3)

(30) (40) (2)

(1)

T(7) = 6 + T(6)

5 + T(5)

4 + T(4)

3 + T(3)

2 + T(2)

Swaps ⇒ $\frac{n-1}{2} \times n) \Rightarrow n^2$

Comparisions: - 1 + 2 + 3 + 4 . . . . n

⇒ $O(n^2)$.

If I

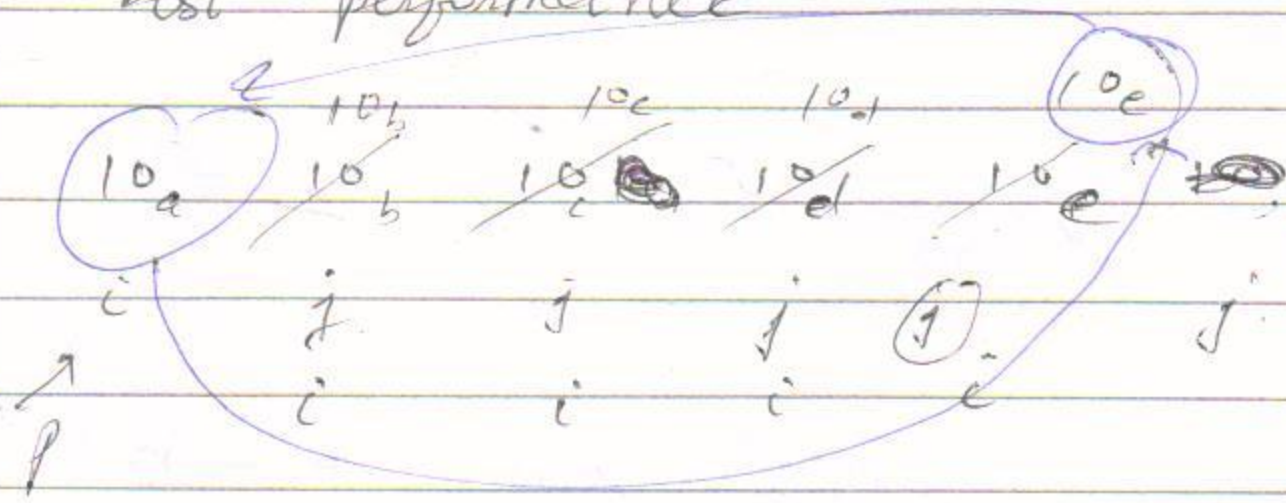
NOTE

Note:-

Date

⇒ If the array is already sorted (ascending or descending) Quick sort algo always give worst case performance.

⇒ If the array is not at all sorted then quick sort gives best performance



⇒ If all elements are same then also Quick sort gives worst performes, and also repeated elements ~~place~~ orders is also changed thus it is not stable ~~sorting~~ method

⇒ Recursive Tree Method:-

$T(n) = 2T(n/2) + n$ ⇒ one recursive term → go for substitution method

① $T(n) = T(n/2) + T(n/2) + n$

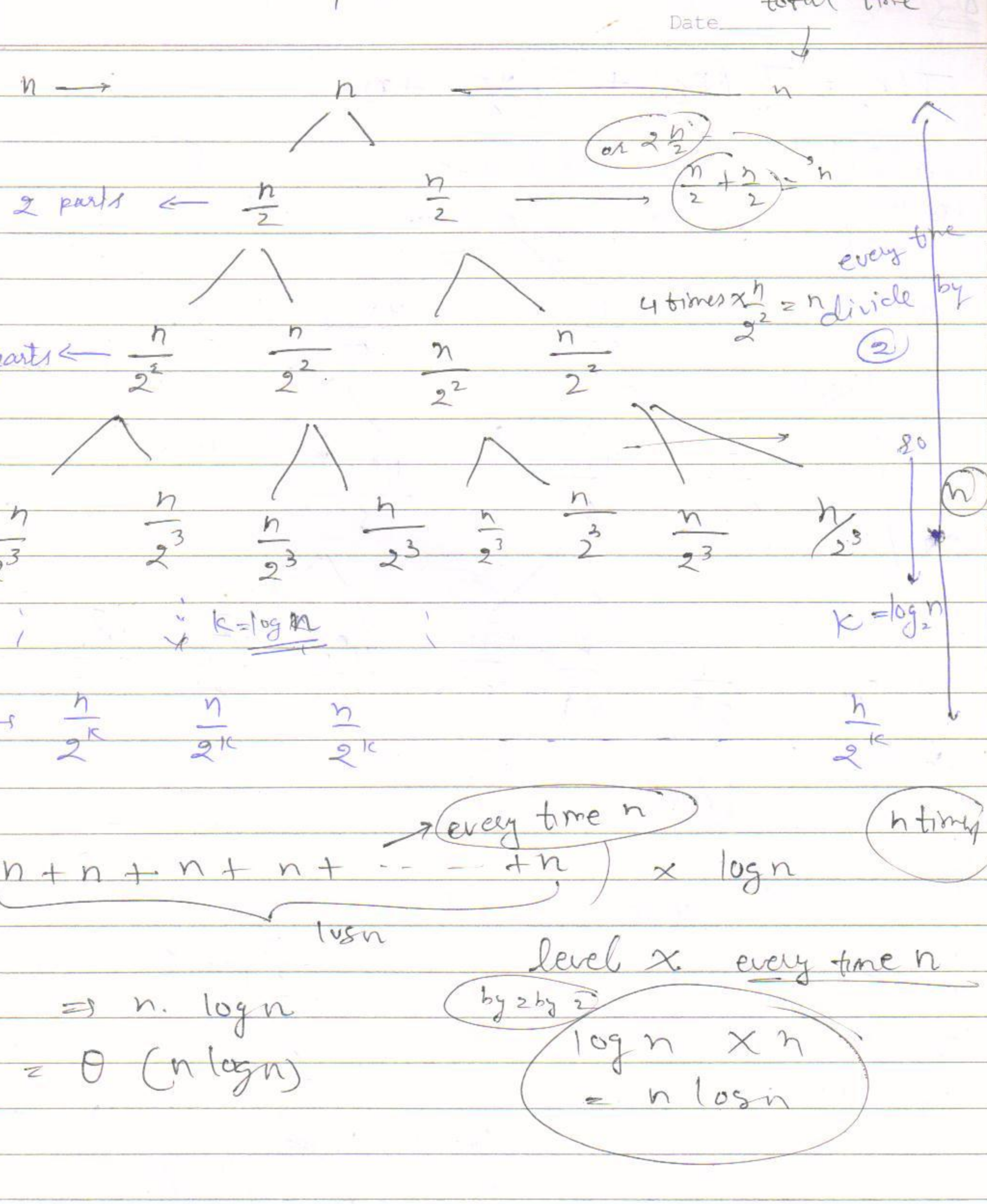
② $T(n) = T(n/5) + T(4n/5) + n$ 2 Recursive term go for R. Tree. Method

Let

①

Date

total time


n
n
n
or 2 n/2
n/2 + n/2 = n
2 parts ← n/2
n/2
every time
4 times x n/2^2 = n divide by
(2)
2^2 parts ← n/2^2 n/2^2 n/2^2 n/2^2
2^3 parts ← n/2^3 n/2^3 n/2^3 n/2^3 n/2^3 n/2^3 n/2^3 n/2^3
so
(n)
∴ k = log n
k = log_2 n
2^k parts → n/2^k n/2^k n/2^k n/2^k
every time n
n + n + n + n + ... + n
log n
x log n
n times
so ⇒ n. log n
= O (n log n)
level x. every time n
by 2 by 2
log n x n
= n log n

Q.2

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + n$$

n ele give to the par

$\frac{1}{5}(n)$ — $\frac{4}{5}(n) = \frac{n}{\frac{5}{4}}$

$\frac{n}{5}$ — $\frac{4n}{5}$ — $\{n\}$

$n \longrightarrow n$

$\frac{n}{5} \longrightarrow \frac{n}{5^1}$ — $\frac{4n}{5} = \frac{n}{(5/4)^1}$ →

$\frac{n}{5^{-2}} \longrightarrow \frac{n}{5^2}$ — $\frac{4n}{5^2}$ — $\frac{4^2 n}{5^2} = \frac{n}{(\frac{5}{4})^2}$ — $\{n\}$

$\frac{n}{5^3}$ — $\frac{4n}{5^3}$ — $\frac{n}{5^3}$ — $\frac{4^2 n}{5^3}$ — $\frac{4^0 n}{5^3}$ — $\frac{4^2 n}{5^3}$ — $\frac{4^2 n}{5^3}$ — $\frac{4^3 n}{5^3}$

$\frac{n}{5^k}$ — gone

$\frac{n}{5^{-k}} = 1$

$\log_5 n = k$

$\log_5 n = k$

$n = 100$

20 — $20\ 20\ 20\ 20\ 20$

$\frac{n}{5}$ — $\frac{4n}{5}$

$\frac{n}{5} + \frac{4n}{5} = \{n\}$

more hight

so

$\log_{\frac{5}{4}} n$

$n \log_5 n$ Lower bound.

k — n — n — n — n

more level

so — less than n

$\frac{n}{(\frac{5}{4})^k} = 1$ — $O(n)$ { $< n$ $< n$ $\}$ — $< n$ $\{n\ (\frac{5}{4})^k$

$n = (\frac{5}{4})^k$

$\log_{\frac{5}{4}} n = k \log_{\frac{5}{4}} \frac{5}{4}$

$\log_{\frac{5}{4}} n = k$

$\log_2 100 : \log_5 100$

$7 : 2.0$

$n = 100$

$\frac{100}{2}$ — $\frac{100}{10}$

100	100 ✓
50	10 ✓
25	1 ✓
12	
6	
3	
2	
1	

$\Rightarrow n + n + n + n + n + \ldots + n$ (× $\log_{\frac{5}{4}} n$)

Upper bound $\Rightarrow O\left(n(\log_{\frac{5}{4}} n)\right)$ always less than this because some level are missing.

Q $T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$

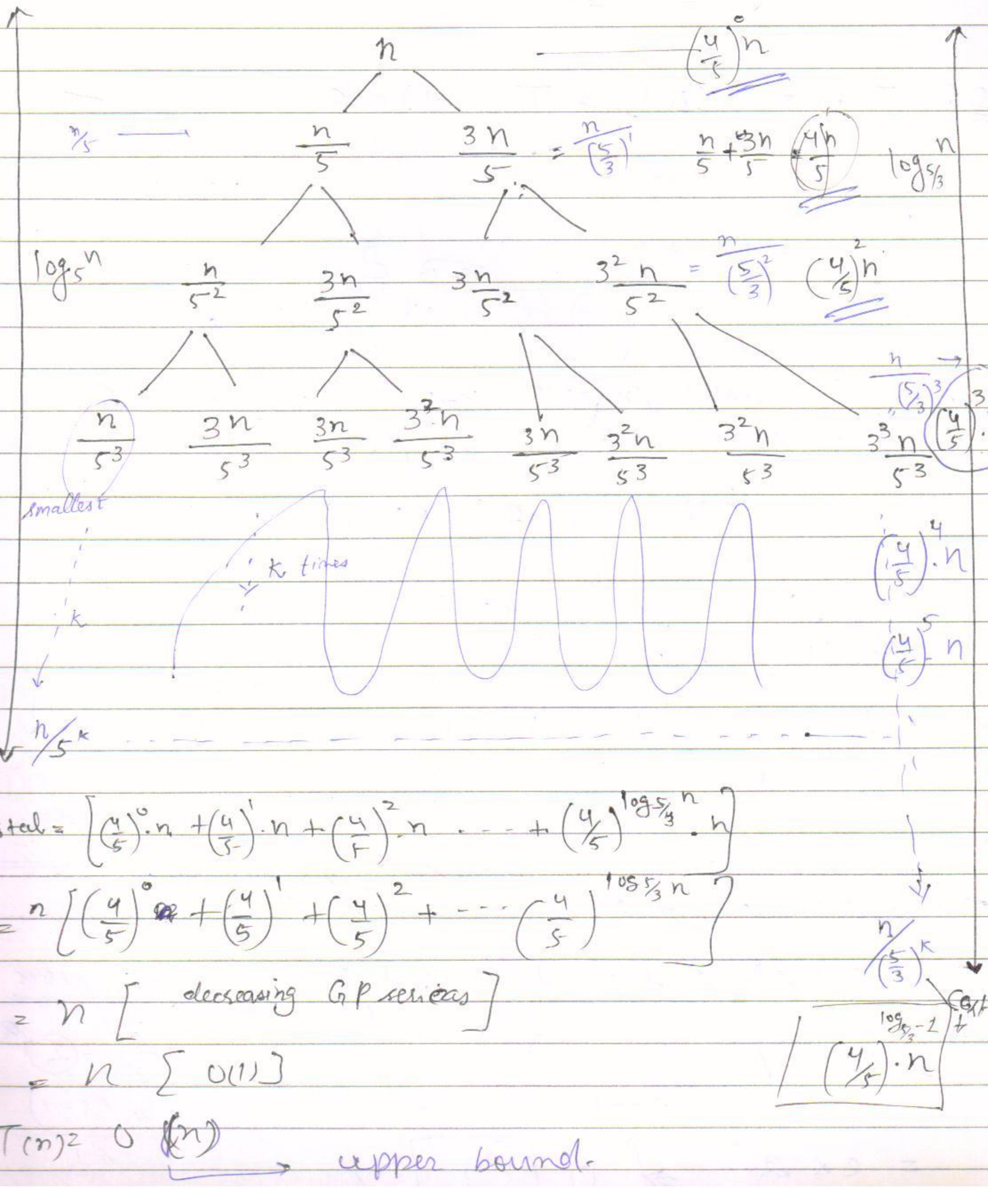
$\Rightarrow O(n \log_{\frac{10}{9}} n)$ A.

Date

$\frac{100}{5}$ $\frac{500}{5}$

20) 20 20 20 20 20

Q.4 $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{3n}{5}\right) + n$



$$\text{Total} = \left[\left(\frac{4}{5}\right)^0 \cdot n + \left(\frac{4}{5}\right)^1 \cdot n + \left(\frac{4}{5}\right)^2 \cdot n \cdots + \left(\frac{4}{5}\right)^{\log_{5/3} n} \cdot n\right]$$

$$= n\left[\left(\frac{4}{5}\right)^0 + \left(\frac{4}{5}\right)^1 + \left(\frac{4}{5}\right)^2 + \cdots \left(\frac{4}{5}\right)^{\log_{5/3} n}\right]$$

$= n$ [decreasing GP series]

$= n$ [$O(1)$]

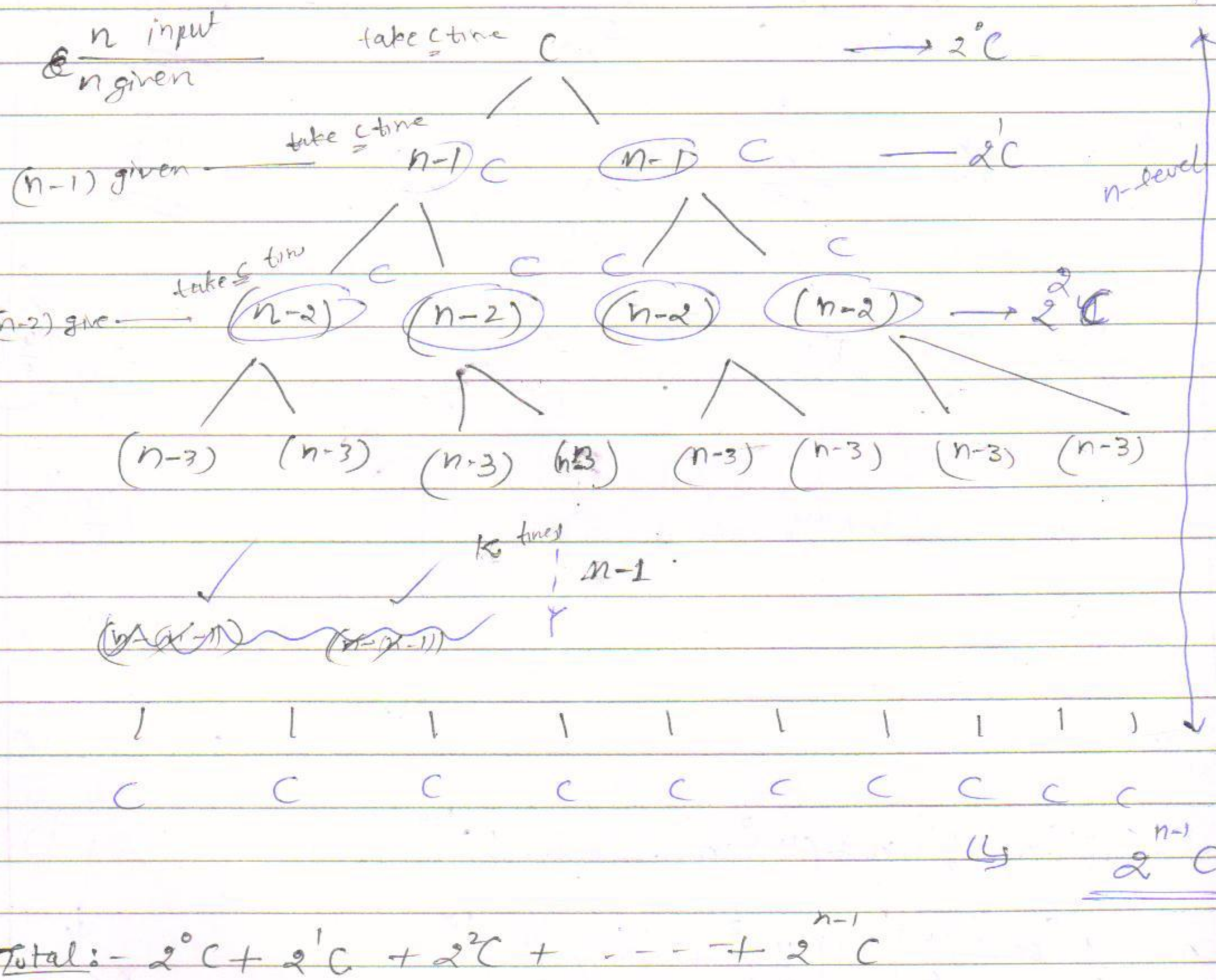
$T(n) = O(n)$ → upper bound.

⑤ $T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{7n}{10}\right) + n$ Date

$\frac{n}{10} + \frac{7n}{10} \Rightarrow \left(\frac{8n}{10}\right)$ → will be decreasing series

so ⇒ Ans. $O(n)$

Q.6 $T(n) = T(n-1) + T(n-1) + c$



Total:- $2^0 C + 2^1 C + 2^2 C + \cdots + 2^{n-1} C$

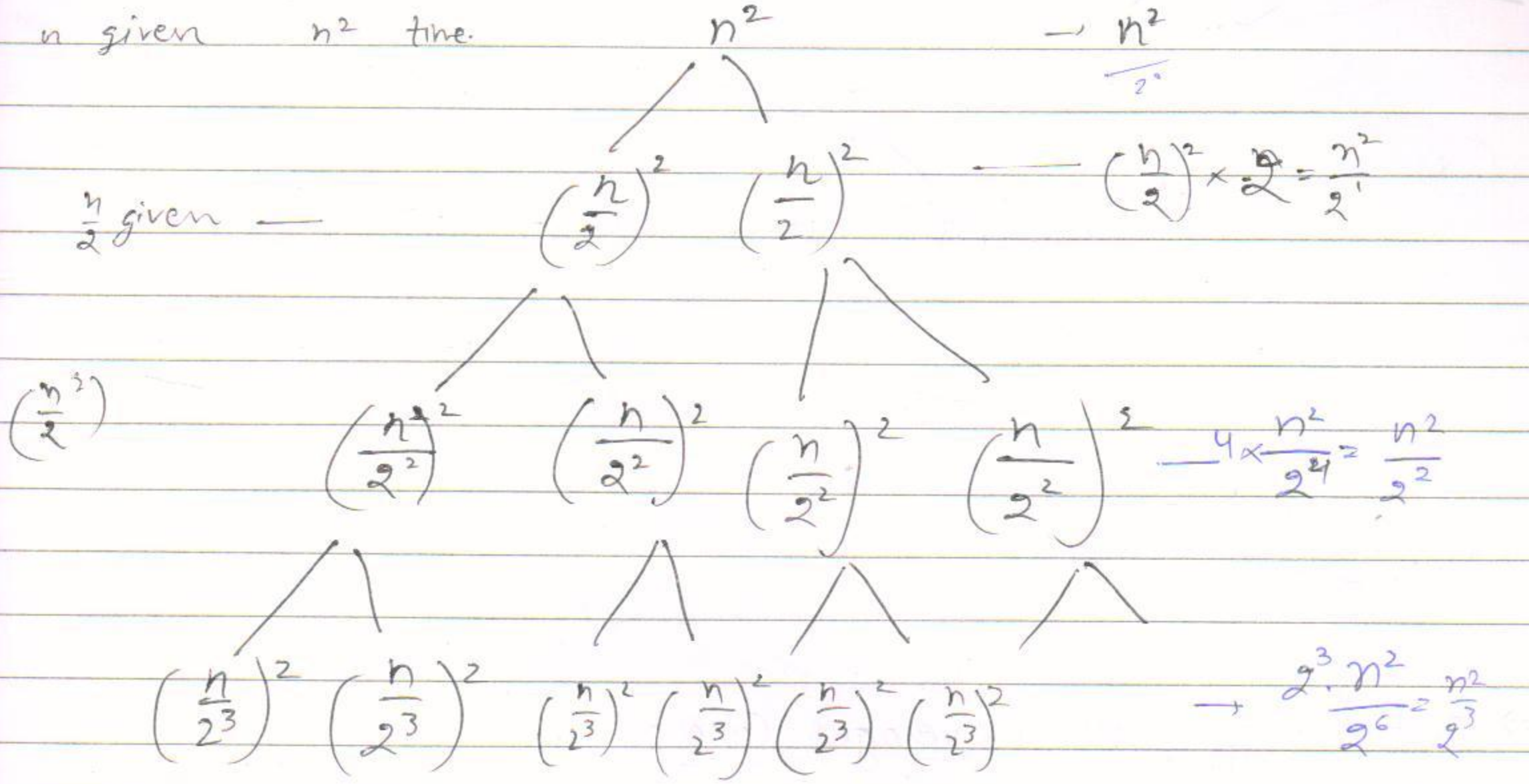
$= C\left[2^0 + 2^1 + 2^2 + \cdots + 2^{n-1}\right)$ GP series

$= C \times \left[1\ \frac{(2^n - 1)}{2-1}\right]$ $\frac{a^2}{a^1} = (a)^R$

$= C \times 2^n \Rightarrow [O(2^n)]$ Ans $\frac{a(r-1)}{r-1}$

(7) $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n^2$

Date



$\therefore$ tot $= \frac{n^2}{2^0} + \frac{n^2}{2^1} + \frac{n^2}{2^2} + \frac{n^2}{2^3} + \cdots + \frac{n^2}{2^{\log n - 1}}$

$= n^2 \left[\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \frac{1}{2^{\log n - 1}}\right]$

$= n^2$ [decreasing gp. series]

$= O(n^2)$ Ans

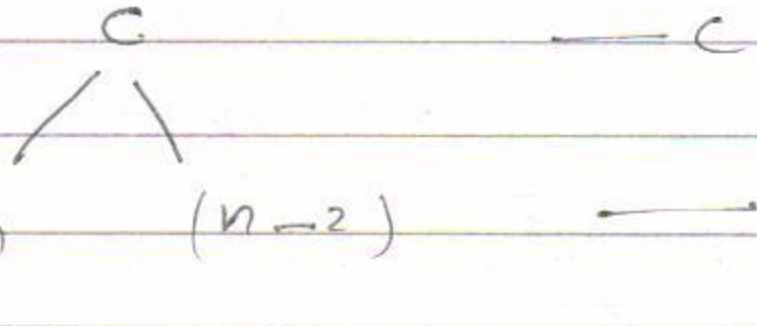
$\frac{n^2}{2^4}$

$\frac{n^2}{2^5}$

$\frac{n^2}{2^{k-1}}$

$\frac{n^2}{2^{\log n - 1}}$

Q8 $T(n) = T(n-1) + T(n-2) + C$ Date



Ans. $O(2^n)$

⇒ Best case && Average Case :-

$$T(n) = T(\alpha.n) + T((1-\alpha).n) + n \Rightarrow O\left[n.\log_{\frac{1}{1-\alpha}} n\right]$$ T.C

where $0 < \alpha < 1$

stack space = max of $\{\log_{\frac{1}{\alpha}} n, \log_{\frac{1}{1-\alpha}} n\}$

$T = T(n/2) + T(n/2) + n \quad \rightarrow n\log_2 n$

$= T(\frac{n}{5}) + T(\frac{4n}{5}) + n \quad \rightarrow n\log_{5/4} n$

$\therefore$ T.C = $O\left[n\log_{\frac{1}{1-\alpha}} n\right]$ or $O\left[n\log_{\frac{1}{\alpha}} n\right]$

stack Space = $\max\left[\log_{\frac{1}{\alpha}} n \text{ or } \log_{\frac{1}{1-\alpha}} n\right]$

$\Rightarrow T(n) = T(n/10) + T(9n/10) + n \Rightarrow n\log_{10/9} n$

Worst case :-

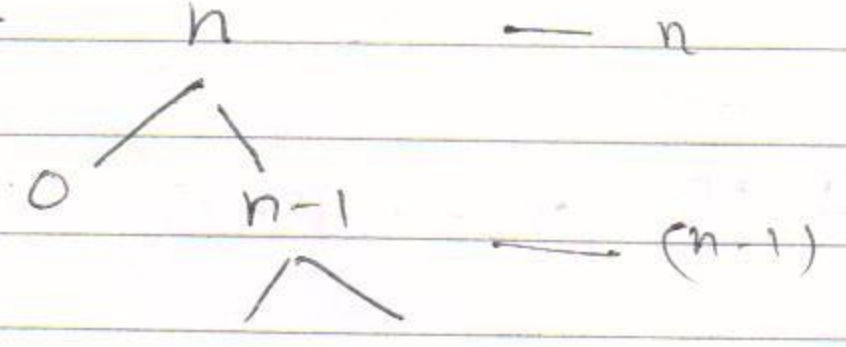
① $T(n) = T(0) + T(n-1) + n$ → $O(n^2)$

→ n level

every time (n)

$\Rightarrow n \times n = n^2$

n ele give — — — n — n



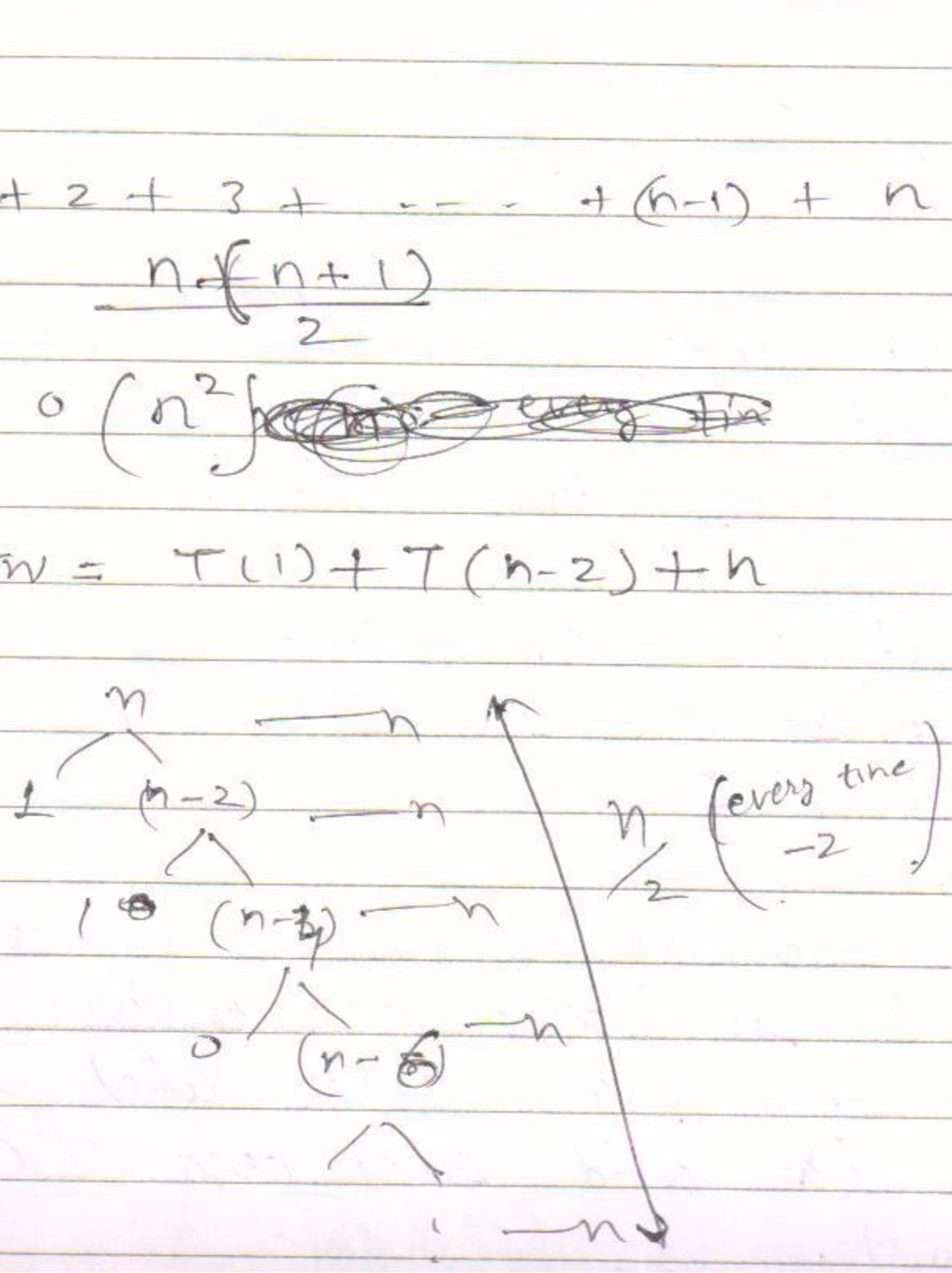
1 (1)

$1 + 2 + 3 + \cdots + (n-1) + n$

$= \frac{n(n+1)}{2}$

$T(1) = O(n^2)$

② $T(n) = T(1) + T(n-2) + n$ $\Rightarrow \frac{n}{2} \times n = \frac{n^2}{2} \Rightarrow O(n^2)$



Q.3 $T(n) = T(3) + T(n-4) + n$ → $\frac{n}{4} \times n \Rightarrow n^2$ Date

∴ $T(99) + T(n-100) + n \Rightarrow \frac{n}{100} \times n = n^2$

⇒ Worst case :—

$$\boxed{T(n) = T(c) + T(n-(c+1)) + n}$$

where c is a constant

$0 \leq c \leq (n-1)$

T.C ⇒ $O(n^2)$

S.C ⇒ $\frac{n}{c+1} \Rightarrow O(n)$ but due to Tail Recursive Method

$\Rightarrow O(\log n)$ ← reduced to

H.W

Q.1 In Quick sort the sorting of n natural nos are selected as the $(n/4)^{th}$ smallest element is selected as a pivot using $O(n)$ time Algorithm then

✓ What is the worst case Time Com of Quick sort

(a) $O(n \log n)$ (b) $O(n^2)$

(c) $O(n)$ (d) $O(n^3)$

Q.2. In quick sort the sorting of n natural nos. the $(n/4)^{th}$ element is select as pivot ele. using ~~$O(n)$~~ $O(n^2)$ time C. algo then what is the ~~Best~~ worst case and Best case T.C. of the quick sort.

Date

$n = 100$

$\frac{n}{4}$ $\quad \frac{100}{4} = 25^{th}$

$\left(\frac{n}{4}\right)$

$\left(\frac{n}{4}\right)$ $\quad \left(\frac{n}{4}\right)$ $\quad \left(1 - \frac{n}{4}\right)$

$\left(\frac{n}{4}\right)$ ① $= \left(\frac{3n}{4}\right)$.

$\left(\frac{n}{4} - 1\right)$

$n \log_{4/3}$ ⑤

$n^2 \longrightarrow n^2$

n

Ans.Q.1 $T(n) = O(1) + O(n) + T(c) + T(n - (c+1))$

40 20 70 50 30 60 10
1 2 3 4 5 6 7

11 pivot

(10 30 20) 40 (70 50 60)

selecting pivot element → partion

$\Rightarrow O(n) + O(n) + T\left(\frac{n}{4} - 1\right) + T\left(n - \frac{n}{4}\right)$

$\Rightarrow O(n) + O(n) + T(n/4) + T\left(\frac{3n}{4}\right)$

$= T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n$

$= O(n \log_{4/3} n)$ $\Rightarrow$ Best case & Worst case is

it is not given smallest $\frac{n}{4}$ th
so anywhere it.

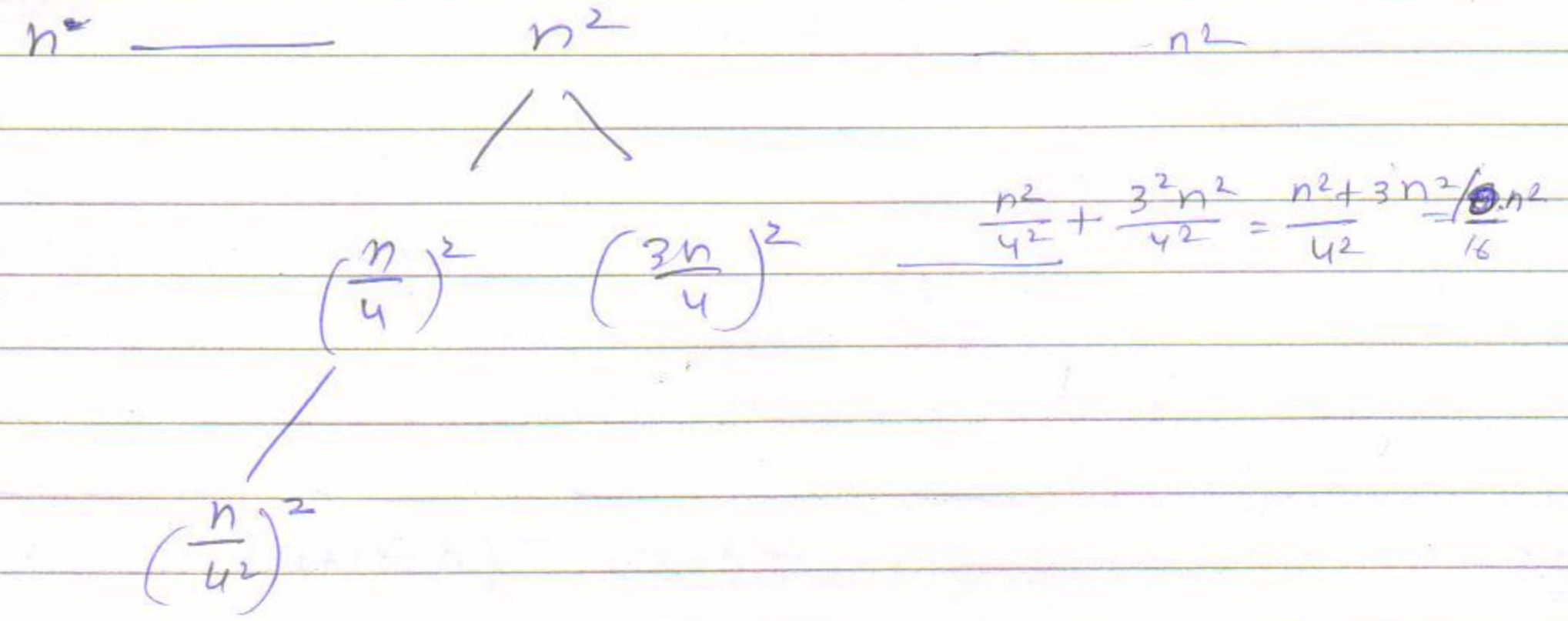
Date

Ans of Q.2 :- → selecting pivot point (elem)

$$T(n) = O(n^2) + O(n) + T\left(\frac{n}{4}-1\right) + T\left(n-\frac{n}{4}\right)$$

$$= n^2 + n + T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right)$$

$$= O(n^2)$$

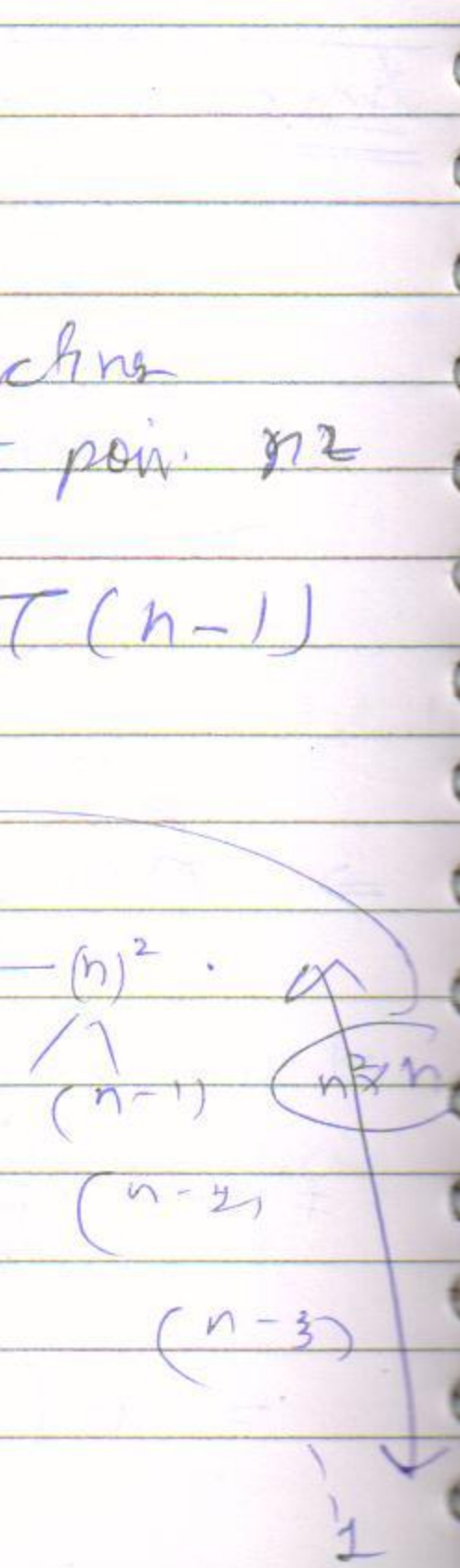


Q. if only $\left(\frac{n}{4}^{th}\right)$ selecting pivot poin. n^2

then $T(n) = O(n^2) + (n) + T(0) + T(n-1)$

$= n^2 + T(n-1)$

$= O(n^3)$



if pivot selecting $\Theta(n)$

Q. n^{th} smallest ⇒ max → worst

⇒ so $O(n^2)$

Q. 1 smallest ⇒ $O(n^2)$

selected as

Q. if Median is a pivot element always then ~~$\left(\frac{n}{2}\right)^{th}$~~ T.C. = ?? ~~smallest element~~, so always half-half.

so → best case = T.C ⇒ $O(n \log n)$

Merge sort	Q. sort
B.C → $n \log n$	$n \log n$ ⇒ $T(n) = 2T(n/2) + n$
W.C → $n \log n$	n^2 ⇒ $T(n) = T(n-1) + n$
Average Case → $n \log n$	$n \log n$ ⇒ $T(n) = 2T(n/2) + n$
$2T(n/2) + n$	When sorted then only quick
$\Theta(n \log n)$	so No need to apply sorting when already sorted so

→ Randomized Quick Sort :-

(1) → Choosing pivot element randomly is called randomised quick sort.

(2) No input behaves worst case performance.

(3) It is independent of input order.

Date

If array is sorted & pivot is 1st element always so worst case of normal quick sort

10 20 30 40 50 60 70

1 2 3 4 5 6 7

p

( ) 10 (20 30 40 50 60 70)

p

( ) 20 (30 40 50 60 70)

p

( ) 30 (40 50 60 70)

( ) 40 (50 60 70)

( ) 50 (60 70)

( ) 60 (70)

Now

Randomised ⇒ Pivot element is changing everytime.

40 10

10 20 30 40 50 60 70

1 2 3 4 5 6 7

r = RG(1,7)

swap(a[p], a[r])

40 20 30 40 50 60 70

i j

(20 30 10) 40 (50 60 70)

r = RG(1,3)

swap(a[p], a[r])

(10) 20 (30)

LLLUUULLL ... ⇒ Lucky.

50

(20 30 10) 40 (50 60 70)

r = RG(5,7)

swap(a[p], a[r])

70 60 50

(60 50) 70 ( )

r = RG(5,6)

50 (60) 70

Algo:- Randomized

Date

```
RQS (a, p, q)
 {
    if ( p==q) return (a[p]);
    else
    {
  | r = RQ(p, q) ; swap (a[p], a[r]);
  m = partition (a, p, q)
      RQS(a, p, m-1);
      RQS ( a, m+1, q) ;
      return (a);
    }
 }
```

due to this line → BC = n log n
WC = n log n
AC = n log n.

In Normal Quick sort:-

Now Randomised Q. Sort:— Date

10_d

10_a 70_b 10_c 10_d 10_e 10_f

1 2 3 4 5 6

k^4 = RG (1, 6) swap (a[p], a[ ])

10_d 10_b 10_c 10_a 10_e 10_f

i j

10_f 10_b 10_c 10_a 10_e 10_d ( )

10_e 10_b 10_c 10_a 10_f ( )

10_a 10_b 10_c 10_e ( )

10_b 10_e 10_a

10_c 10_b ( )

Note :- Randomised Quick sort algo will give worst case performance $O(n^2)$ always if all elements are same.

One More incident of worst case :- Randomised Quick sort will give ~~one~~ worst case performance some time although array is not sorted

20 60 40 10 50 30 70

4 k = RG (1, 7)

4 : ( ) 10 (20 60 40 50 30 70)

7 : 10 (20 60 40 50 30) 70 ( )

2 : (10) (20) (60 40 50 30) (70)

6 : (10) (20) (30) (50 40 50) (70)

if random Generator always generates pivot element either smallest element or greatest element then also Worst case see Ex

Quick sort

Date

1. Best case — $n \log n$
2. Average Case — $n \log n$
3. Worst case ⇒ n^2 —— RQS

sorted
↓
99 x

↓
① BC → $n \log n$
② AC → $n \log n$
③ WC → n^2

↓
all ele same.
but this case 99.999 % Not happens.

5.
⇒ Selection - Procedure :-
i/p : An array of n-ele and integer k.
o/p : return k^{th} smallest ele.

Ex:(i) A[40 20 70 50 60 30 10] k=4
1 2 3 4 5 6 7

o/p = 40 Any. k=4
if k=7 = 70

Algo(1) (i) sort the array —— $n \log n$
② return (a[k]) → —— O(1)
—— $\Theta(n \log n)$

Algo(2) : ① Apply k passes of selection sort = $O(kn)$
② return (a[k]) = —— O(1)
—— $\Theta(kn)$

Date

Algo (3) :-

using partition algo

A [40 20 70 50 60 30 10] k=4
  1 2 3 4 5 6 7

p ↑

m =

Algo (3) ~~partition(~~
selection-procedure (a, p, q, k)
{
if (p=q)
{
if (p==k)
return (a[p]);
else
return (-1);
}
else
{
m = partition (a, p, q); → O(n)
if (m==k) return (a[m]); → O(1) — best case if return in first
else
if (k > m) → O(1)
selection-~~partition~~ procedure (a , m+1 , q, k) → T( )
else
selection-~~partition~~ procedure (a, p, m-1, k) T( )
}

Date

Let T(n) be the time complexity of selection procedure on n - elements then —

Best case and Average case Partion

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ O(n) + O(1) + O(1) + T(n/2) & \text{if } n > 1 \end{cases}$$

$T(n) = O(n) + T(n/2)$

IMP

Worst case Partition — just name sake

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ O(n) + O(1) + O(1) + T(n-1) & \text{if } > n \end{cases}$$

$$T(n) = T(n-1) + n$$

$$= O(n^2)$$ in case if array already sorted.

Stack space ⇒ level of the

Space Comp = O(n)

but = O(log n)

reduced to

Best case

$T(n) = T(n/2) + n$

$= n + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^{\log n}}$

$= n \left\{ \frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \right.$

$= n [O(1)]$

$T(n) = O(n)$

Space Com = O(log n)

⇒ Strassen's Matrix Multiplication:—

Date

→ without ∑ A C.

Matrix - Addition

$A_{2\times2}$, $B_{2\times2}$

$C = A + B$

↓

2×2

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ $B = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$

$C = \begin{bmatrix} 3 & 5 \\ 7 & 9 \end{bmatrix}$ $\Rightarrow 2^2$ ele

$\Rightarrow 2^2 * 1 \Rightarrow O(2^2)$

every ele require 1 addition

so

$\Rightarrow n^2$ — ele

n^2 — 1 addition

$\Rightarrow O(n^2)$

ex. $A_{10\times10}$ $B_{10\times10}$

$C = A + B$

↓

$n_0 = 10^2$

so $O(10^2)$

Add $= O(n^2)$

Matrix Multiplication

$A_{2\times3}$ $B_{3\times2}$

$C = AB$

↓

2×2

$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

$C = \begin{bmatrix} \text{3 multi} & \text{3 multi} \\ \text{3 mul} & \text{3 multi} \end{bmatrix}$ $\Rightarrow 2^2$ elel

$\Rightarrow 2^2 * 3$ multiplica

#elem * evaluting 3 ele multiplicatn

$2^2 * 3 =$

$\Rightarrow 2 * 2 * 3 \Rightarrow 12$

$A_{m\times n}$ $B_{n\times p}$

$m \times p$

$\Longrightarrow m * p * n$

$= O(m m p)$

$= O(n . m . n)$

Mul $= O(n^3)$

Addition :-

```
for( i = 1 to n)
  for( j = 1 to n)
```

} 2 for loops.

Multiplication :-

3 for loops

⇒ Using DAC :- - Strassen's Matrix Multiplication

⇒ Matrix - Multiplication :-

means → matrix size is (2x2)

① if n is {2x2} or less than {2x2} then the problem is small.

② n should be power of 2

③ Matrix should be Square matrix.

Ex :-

$$A = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad 3\times3 \quad 4\times4 \checkmark$$

$$B = \begin{bmatrix} 2 & 3 & 4 & 0 \\ 5 & 6 & 7 & 0 \\ 8 & 9 & 10 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad 3\times3 \quad 4\times4 \checkmark$$

$A_{2\times2}$ $B_{2\times2}$

4x(4) (4)x4

16x(16) (16)x16

16x(16) (8)x8

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad 4\times4 \xrightarrow{2} 2\times2$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad B = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \quad 4\times4 \xrightarrow{2} 2\times2$$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

one multiplication ↑ 4 additions ↓

1 146

Date

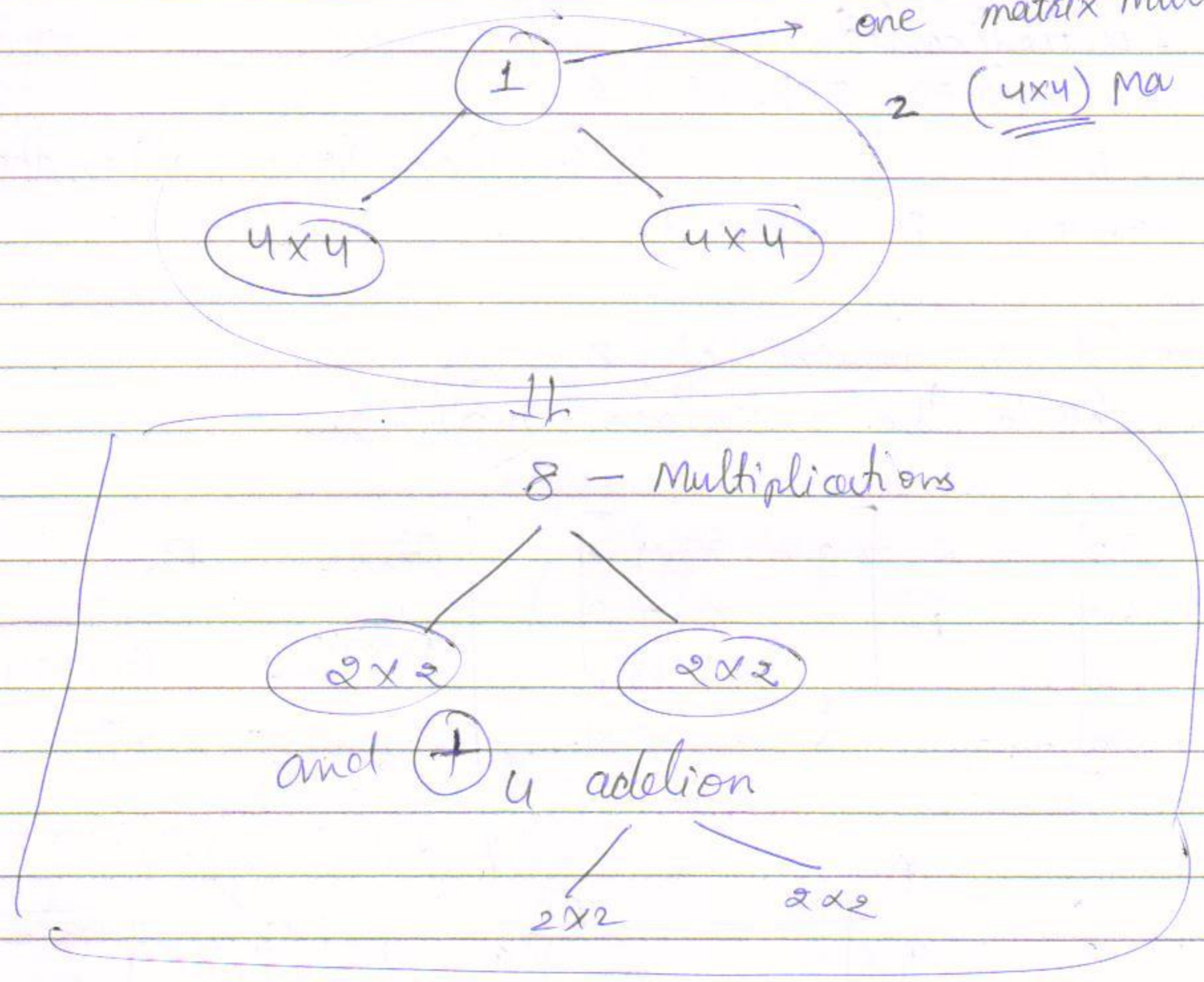
$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$$

$$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$$

$$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$$

$$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$$

} 8 multiplication



without any addition of 2 Matrix.

$$\Rightarrow \begin{cases} n \times n = n^2 \end{cases}$$

$$T(4) = 8\,T\left(\frac{4}{2}\right) + 4 * \left(\frac{4}{2}\right)^2$$

$$T(8) = 8\,T\left(\frac{8}{2}\right) + 4 * \left(\frac{8}{2}\right)^2$$

$$T(64) = 8\,T\left(\frac{64}{2}\right) + 4 * \left(\frac{64}{2}\right)^2$$

$$\boxed{T(n) = 8\,T\left(\frac{n}{2}\right) + 4 * \left(\frac{n}{2}\right)^2}$$

Date

Let $T(n)$ be the amount of time required to multiply 2 matrices of size $n \times n$ then—

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 2 \times 2 \\ 8\,T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2 & \text{if } n > (2 \times 2) \end{cases}$$

$$T(n) = 8\,T\left(\frac{n}{2}\right) + \frac{4\,n^2}{2^2}$$

$$T(n) = 8\,T\left(\frac{n}{2}\right) + n^2$$ ——— combine time & sub problem

↳ size of subproblem.

$$T(n) = O(n^3)$$

stack space $= O(\log n)$

↳ every time divide by 2.

NOTE:

Both Using divide and conquer & Without using DAC matrix multiplication ~~time~~ takes same time $O(n^3)$. so Without DAC is best one because ~~without~~ ~~takes~~ with DAC takes $\log n$ extra space.

According to Strassen's.

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 2 \times 2 \\ 7\,T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 & \text{if } n > 2 \times 2 \end{cases}$$

$$\rightarrow 7\,T\left(\frac{n}{2}\right) + \frac{18}{4}\left(n^2\right) \rightarrow 7\,T\left(\frac{n}{2}\right) + 4.5\,n^2$$

$\Rightarrow 7T(n/2) + 4.5n^2$

Date

$T(n) \Rightarrow 7T(\frac{n}{2}) + n^2$

$\Rightarrow O(n^{2.81})$

$\Rightarrow O(n^{2.81})$

space complexity
$\Rightarrow O(\log n)$

> Note :- A new Algo recently came in market that takes $O(n^{2.32})$ time

$\Rightarrow$ Master Theorm :-

$$T(n) = aT(n/b) + f(n)$$

$$a \geq 1 \quad , \quad b \geq 2$$

Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ where ϵ = a constant & $\epsilon > 0$

then

$$T(n) = \Theta(n^{\log_b a})$$

Case 2: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ where ϵ = a constant $\epsilon > 0$

then

$$T(n) = \Theta(f(n))$$

Case 3: If $f(n) = \Theta(n^{\log_b a})$ → take one person from $f(n)$ & $n^{\log_b a}$ and multiply by $\log n$

then

$$T(n) = \Theta(f(n) \cdot \log n)$$

Ex(i) : $T(n) = 8T\left(\frac{n}{2}\right) + n^2$ Date

given : $a = 8$

$b = 2$

$f(n) = n^3$

$n^{\log_b a} = n^{\log_2 8} \Rightarrow n^{\log_2 2^3} = n^3$ (3)

$f(n) = O\left(n^{\log_b a}\right)$

$n^2 = n^3$

so case (1) $\theta\left(n^{\log_b a}\right) = \theta(n^3)$

Ex(2) :- $T(n) = 2T(n/2) + n^2$

$a = 2$ $f(n) = n^2$

$b = 2$

$n^{\log_b a} = n^{\log_2 2} = n$

Ans. $\theta(n^2)$

Ex3 : $T(n) = 2T(n/2) + n$

$a = 2$

$b = 2$ → f(n)

$n^{\log_2 2} \Rightarrow n$ same $\therefore \theta(n \cdot \log n)$

Ex. 4 :- $T(n) = 4T(n/2) + n^3$

$a = 4, b = 2$ $f(n) = n^3$

$n^{\log_2 4} = n^2$ Ans. $\theta(n^3)$.

Date

Case 1:- Comparing $n^{\log_b a}$ and $f(n)$, $n^{\log_b a}$ is ~~great~~ polynomial times greater than $f(n)$.
(polynomial times mean n^{ϵ} time greater.)

Case 2: Comparing $n^{\log_b a}$ and $f(n)$, $n^{\log_b a}$ is polynomial time smaller than $f(n)$.
or. ($n^{\log_b a}$ is n^{ϵ} times smaller).

Case 3:- ~~Comparing~~ $n^{\log_b a}$ and $f(n)$, asymptotically equal.

Ex. 5:- $T(n) = 2T(n/2) + n \log n$

$a = 2$, $b = 2$

$n^{\log_b a} \Rightarrow n^{\log_2 2} \Rightarrow (n)$ → Compare.

$f(n) = n \log n$

$n < n \log n$

But it is $(n < n \log n)$

Comparing $n^{\log_b a}$ and $f(n)$, $f(n)$ is greater by logarithmic times not polynomial times. so masters theorm is not applicable. and go for substitution method.

Q. $T(n) = T(\sqrt{n}) + c$

A $= 1 \cdot T\left(\frac{n}{1}\right)^{1/2} + c$

$= a = 1$, $b = 1$ → (masters theorm not applicable)

(b) should be greater then 1 $(b > 1)$

Date

Now to solve it assume.

① assume $n = 2^k$

before $T(n) = T(n^{1/2}) + c$

after $T(2^k) = T(2^{k/2}) + c$ ——— power in terms of k and rest in term of S.

② assume $T(2^k) = S(k)$

$T(2^k) = T(2^{k/2}) + c$ ✓

$\Rightarrow S(k) = S(\frac{k}{2}) + c$

Now master theorm — $a = 1$, $b = 2$

$k^{\log_b a} \Rightarrow k^{\log_2 1} \Rightarrow k^0 = 1$ Am I = C

$= C$ ——— Compare

$S(k) = \theta(c \cdot \log k)$

No replacement $\Rightarrow$ $T(2^k) = \theta(c \cdot \log k)$

$T(2^{\log_2 n}) = \theta(c \cdot \log(\log n))$ ✓

Q $T(n) = 2T(\sqrt{n}) + \log n$

① assume $n = 2^k$

$\log n = k \log 2$

$n = 2^k$

$\log n = k \log 2$

$T(n) = 2T(2^{k/2}) + k$

② assume —

$T(2^{(k)}) = S(k)$

$\Rightarrow S(k) = 2S(k/2) + k$

* Recurrence

Date

$S(k) = \theta(k \cdot \log k)$

③ $T(2^k) = \theta(k \cdot \log k)$

④ $T(2^{\log_2 n}) = \theta(\log n \cdot \log \log n)$

$T(n) = \theta(\log n \cdot (\log \log n))$

Gate

Q. A(n)
{
if ($n \leq 2$) return (1); ——— O(1) if = $n = 2$
else
return (A($\sqrt{n}$))} ⇒ or else ⇒ O(1)
{m = $\sqrt{n}$; ——— A($n^{1/2}$)
b = A(m); ——— T($\sqrt{n}$)
return (b);
}

T.C. = ?

(a) O(n) × (b) O(log n) (c) O(log(log n)) ✓ (d) O(1)

A. A(1000) ✓ method 1 ⇒ method 2 ×

A(33) ✓
A(5) ✓
A(2)

1000 → 3

(1) Write Recurrence relation.

$$T(n) = \begin{cases} O(1) & n \leq 2 \\ T(n^{1/2}) + c & n > 2 \end{cases}$$

Recursive term — Non recursive term. (cost of divide & combine step)

2^{16} 2^8 2^4 2^2 2 — 16

Date

$$T(n) = T(\sqrt{n}) + c$$

① assume $\longrightarrow$ $n = 2^k$

$$T(2^k) = T(2^{k/2}) + c$$

② $T(2^k) = S(k)$

$$S(k) = S(k/2) + c$$

$a = 1 \quad b = 2$

$k^{\log_b a} \Rightarrow k^{\log_2 1} \Rightarrow k^0 \quad f(k) = c$

$\Rightarrow c$

$S(k) = \theta(c \cdot \log k)$

$T(2^k) = \theta(\log k)$

$T(2^{\log n}) = \theta(\log \log n)$

$T(n) = \theta(\log \log n)$

A.

Gate Q. T.C = ?

```
A(n)
{
  if (n <= 1) return (n)          O(1)
  else
  { return ( A(n/2) + A(n/2) + n )
  }
}
```

(function) (function) (Not function)

(a) O(n) (b) O(n log n) (c) O(n²) (d) O(log n)

Ans.

$$T(n) = \begin{cases} O(1) & n = 1 \\ 2T(n/2) + C & \text{if } n > 1 \end{cases}$$

```
else
{ return (A(n/2) + A(n/2) + n) ⇒
}
```

```
else
{
  b = A(n/2);   → T(n/2)
  c = A(n/2);   → T(n/2)
  d = b + c + n;   O(1)
  return (d);
}
```

Master theorem :—

$a = 2 \quad b = 2 \quad f(n) = c$

$n^{\log_2 2} = n$ ——— larger

$O(n)$

n/2 = partition(a, n)

a = A(a, n/2) T(n/2)

b = A(a, n/2) T(n/2)

do own your own ⟶

Date

⇒ return (A(n/2) + T(n/2) + n²)

→ (+ P(n))

→ (+ BS(n))

→ (+ n)

→ (+ C) →

→ return (n²+n+1); → O(1)

Gate TC=?

Q. A(n)

{

if (n ≤ 1) return (n²) = O(1)

else

{

return (n * A(n/2) − A(n/2));

↑ f ↑ fn

}

(a) O(n) (b) O(n log n) (c) O(n²) (d) O(log n)

Ans.

else

{

a = A(n/2); ⟶ T(n/2)

b = A(n/2); ⟶ T(n/2)

c = (n * a * b); ⟶ constant → O(1)

$$T(n) = \begin{cases} O(1) & n = 1 \\ 2T(n/2) + c & n > 1 \end{cases}$$

a = 2

b = 2

$\log_2^2$

if. changes like this Date

return(n * A(n/2) - A(n/2)); → same ← O(n)

ret (n * A(n/2) * A(n/2)); → same

r (n * A(n/2) / A(n/2) ; → same

Gate

Q. TC = ?

```
A (n)
{
  if (n ≤ 1) return (n);
  else
  {
    return( 5 * A(n/2) + 3 * A(n/2) + n^2);
  }
}
```

TC = O(n) Ans.

Ans.

```
else
{
  a = A (n/2);   ← T(n/2)
  b = A (n/2);   → T(n/2)
  c = 5*a + 3*b + n^2;   → O(1)
  return (c);
}
```

$$T(n) = \begin{cases} c(1) & n = 1 \\ 2T(n/2) + c & n > 1 \end{cases}$$

M.T

a = 2

b = 2

f(n) = c

$n^{\log_2 2} = n$ ← Ans

O(n)

157

A (n)
{
A (n/2)
A (n/2)
A (n/2)
A (n/2)
}

A (n)
{
5 * A (n/2) ;
}

$T(n) = T(n/2) + C_0$

$T(n) = 5T(n/2) + C_0$

$\Rightarrow 1 + 2 + 3 + 4 + \cdots + n = \frac{n(n+1)}{2} = O(n^2)$

```
for( i=1 ; i<= n ; i++)
    s = s+i
```

$\Rightarrow O(n)$

$TC = O(n)$

so upper bound

$\sum_{i=1}^{n} i = O(n^2)$

$TC = O(n)$

② $\Rightarrow n \cdot n \cdot n \cdot n \cdot n \cdots n \Rightarrow O(n^n)$ upper bound

```
for( i=1 to n)
    s = s*n ;
```

$TC = O(n)$

upper bound $\prod_{i=1}^{n} n = O(n^n)$

$TC = O(n)$

$n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots 1 = n! = O(n^n)$

```
for (i=1 to n)
    s = s*i
```

so $TC \leq n!$ is $O(n)$

Date

$\Rightarrow$ $n^3 + n^2 + n + 1 = O(n^3)$ upper bond

$\Downarrow$

3 sum only $\rightarrow$ Order of 1 $\Rightarrow O(1)$

$\Rightarrow n + n + n + n = O(n)$

4 Additions $\Rightarrow O(1)$

$\Rightarrow T(n) = T(n-1) * n$

Q. Consider the following c program

```
fact (n)
{
 if (n<=1) return (1);
 else
  {
   return ( n * fact (n-1));
  }
}
```

TC =? $\Rightarrow O(n)$

```
else
{
 a = fact (n-1);   ---> T(n-1)
 b = n * a;        ---> O(1)
 return (b);
}
```

$T(n) = \begin{cases} O(1) & n=1 \\ T(n-1)+C & \end{cases}$

$\Downarrow$

$= O(C \cdot n)$

$= O(n)$ $\Rightarrow$

$n \times f(n-1)$

1 1 3 3 3

Date

2 Type of Recurance Relations.

① Value Recurs.ⁿ Relation.

② Time Com. Recurance Relation → According to
↳ (Time complexity Recurrance Relation) ~~vedo~~ Algo.

Fibnucci series →

Value Relation → $T(n) = T(n-1) + T(n-2)$

T.C Relation → $T(n) = T(n-1) + T(n-2) + c$.

Q

```
Mul (a, n)
{
 if (a==0 (or) b==0)
   return (0)
   else
   {
   return ( a + Mul.(a, n-1)) ;
   }
}
```

T.C. ⇒ $T(n) = T(n-1) + c$

& value $T(n) = a + T(n-1)$ ⇒ write as it is.

---

close chapter DAC

8.

⇒ Selection sort :-

Date

ex(1) :-

i/p :- 75 15 40 5 62 1 25 10

1 2 3 4 5 6 7 8

Pass ① :- i = 1 , min = ~~1~~ ~~2~~ ~~4~~ 6 swap(a[i], a[min])

↳ one swap

1 15 40 5 62 75 2 10 (n-1) comparisons

1 2 3 4 5 6 7 8

pass ② :- i = 2

min = ~~2~~ 4 swap(a[i], a[min])

1 5 40 15 62 75 25 10

1 2 3 4 5 6 7 8

Com ⇒ n-2

swaps ⇒ 1

pass ③ :- i = 3

min = ~~3~~ 4 8

1 5 10 15 62 75 25 40

1 2 3 4 5 6 7 8

Com ⇒ n-3

s ⇒ 1

pass ④ :-

i = 4

min = 4

1 5 10 15 62 75 25 40

1 2 3 4 5 6 7 8

pass ⑤ :- i = 5

min 5

1 5 10 15 25 75 62 40

pass (6) :- i = 6

min = 6

1 5 10 15 25 40 62 75

i = 7 min = 7

Date

NOTE:-

(i) In selection sort (n-1) passes and in every pass maximum n comparisions and exactly 1 swap is required. so total comparision are $O(n^2)$. &&
total no. of swaps $\Rightarrow (n-1)$

(ii) Selection sort is in-place and stable.

(iii) It doesn't matter array is sorted or not.

Time Complexity of selection sort is always $\Rightarrow$ $\Theta(n^2)$.

$\Rightarrow$ Insertion Sort:-

i/p: 10, 20, 30, 40, 35, 25, 15, 50, 60, 70, 5

pass (i):- [10]

Pass (ii):- [10 | 20]

Pass iii:- [10 | 20 | 30] — sorted array; compare

Pass iv:- [10 | 20 | 30 | 40] — compare → one time Right shift.

(5):- [10 | 20 | 30 | 40 | 35] → [10 | 20 | 30 | 35 | 40]

3 failure so 3 Right shifts.

(6) [10 | 20 | 30 | 35 | 40 | 25] →

x = 25

[10 | 20 | 25 | 30 | 35 | 40]

Date

(7) | 10 | 20 | 25 | 30 | 35 | 40 | 15 | → | 10 | 15 | 20 | 25 | 30 | 35 | 40 |

x = 15

(8) | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 50 |

yes

(9) | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 50 | 60 |

yes

(10) | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 50 | 60 | 70 |

yes

(11) | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 50 | 60 | 70 | 5 |

10 failure
10 Right shift.

| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 50 | 60 | 70 |

Best Case [Ascending orders] :-

ex:- 10 20 30 40 50

		Comparision	Swap
1.	10	0	0
(2)	10 | 20	1	0
(3)	10 | 20 | 30	1	0
	10 | 20 | 30 | ... | n	1	0
		1	0
		(n-1)	0

{Constant elements are not sorted some}

Note:- Date

① If array is *almost sorted then insertion sort is best.

② If the array contains n elements, to sort the array (n-1) comparision are required in the best case.

③ If ~~arra~~ for smaller size arrays insertion sort is best sorting algo { Merge sort is not advisable. }

⇒ Worst Case :- { In Descending order :- } :

50, 40, 30, 20, 10

		Com	Swap
1.	[50] →	0	0
2.	[50 | 40] ⇒ [40 | 50] →	1	1
③	[40 | 50 | 30] ⇒ [30 | 40 | 50] →	2	2
④	[30 | 40 | 50 | 20] → [20 | 30 | 40 | 50]	3	3
	.	.	.
⑤	[20 | 30 | 40 | 50 | 10] ⇒ [10 | 20 | 30 | 40 | 50]	n-1	n-1
		$\frac{(n-1)n}{2}$	$\frac{n(n-1)}{2}$
		$= n^2$	$\frac{1}{2} n^2$

$n^2 + n^2 = 2n^2$

Average Case :- Date

half time best Case & half time wrost case

1 2 3 4 5 6 7 ---- n

0 1 n 1 n 1 n ....

$\frac{n}{2} \times 1 \Rightarrow \frac{n}{2}$

$\frac{n}{2} \times n = \frac{n^2}{2}$

$O(n^2)$

T.C. $\Rightarrow O(n^2)$

Q. Gate want to add one more element 5)

| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 5 |

insertion order in Last

$n = 5$

7 Right shift

① using L.S. $\Rightarrow$ n - Comparision $\times n \Rightarrow n^2$

TC $\Rightarrow O(n)$ $\Rightarrow$ n swaps. $\times n \Rightarrow n^2$

1 element $\frac{2n^2}{}$

$\Rightarrow O[n^2]$

if n element have to insert then

inserting n elem

② using binary search

| 10 | 20 | 30 | 40 | 50 | 60 | 70 |

1 2 3 4 5 6 7

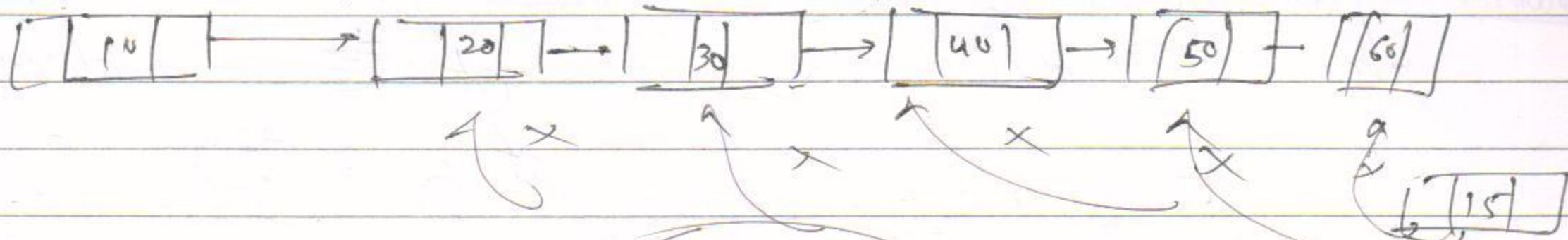
using B.S. $\Rightarrow$ log n Comparision $\times n$ elem $\Rightarrow n \log n$

n swaps $\times n \Rightarrow n^2$ sw

$O(\log n)$ 1 elem $O(n \log n + n^2)$

Date

⇒ Heap Sort :-



using Leaner Search ⇒ n - comparision → $n^2 - c$

1 elem → 0 - swapps → 0 - 5 → n elem

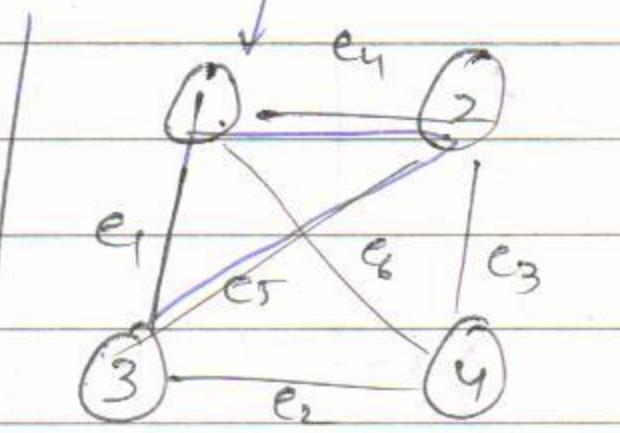
for n elem = $O(n^2)$

✓ we can not apply B.S. in link list.

Thus finally insertion sort Time complexity
is in Best Case → $O(n)$
& Worst case and average case = $O(n^2)$.

⇒ Heap sort :-

Tree	Graph
① Root - element	① No - Root element.
② No - cycle	② Cycle may be or may not be.
③ Tree is always connected graph.	③ It may be connected or disconnected.
④ Directed	④ directed graph (or) Un-directed.
So called ⑤ DAG → Directed Acyclic Graph.	



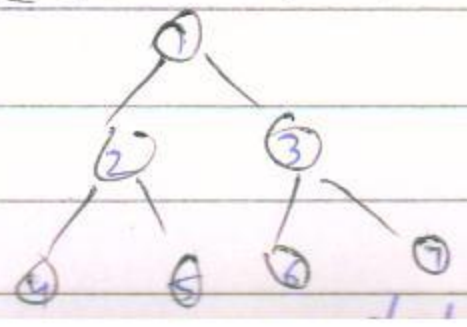
G (V, E)

set of vertices

set of edges.

V = { 1, 2, 3, 4 }

E = { e1, e2, e3, e4, e5, e6 }



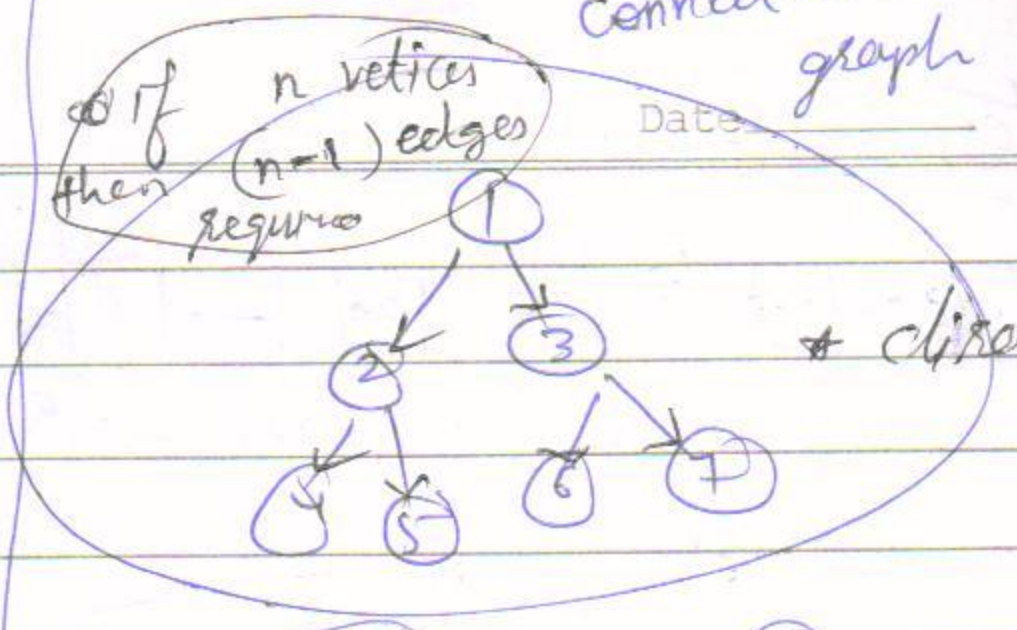
Root :- from where we can reach at all vertices.

Vertices = n
then (1) if # edges = (n-1)
(1) ——→ then ⇒ NO cycle tree

(2) if # edges = n
graph having cycle

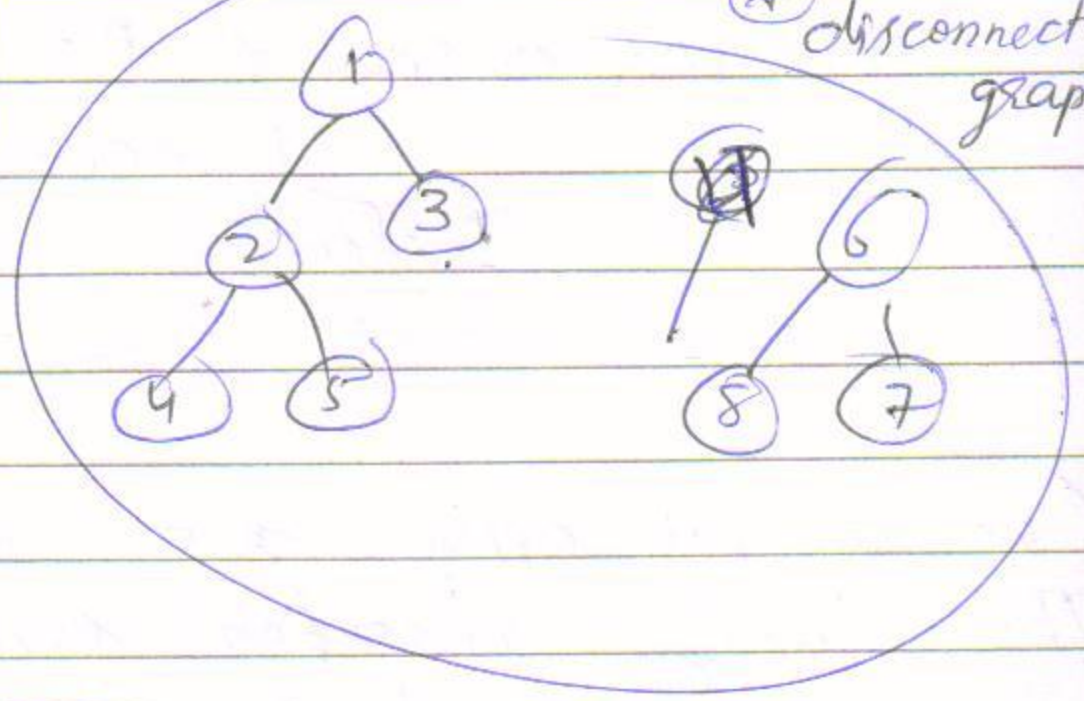
(3) if # of edges = (n-2)
→ disjoint graph

* Tree is always connected graph

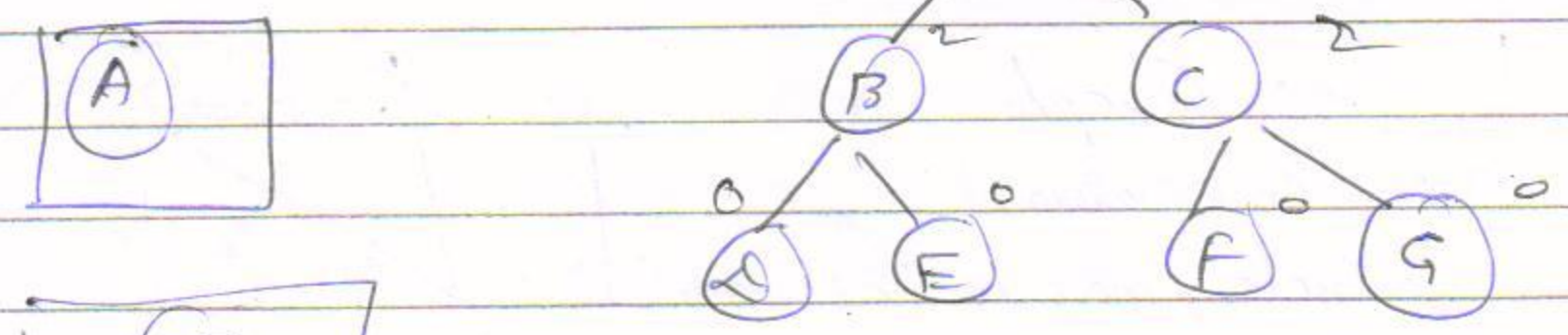


* directed

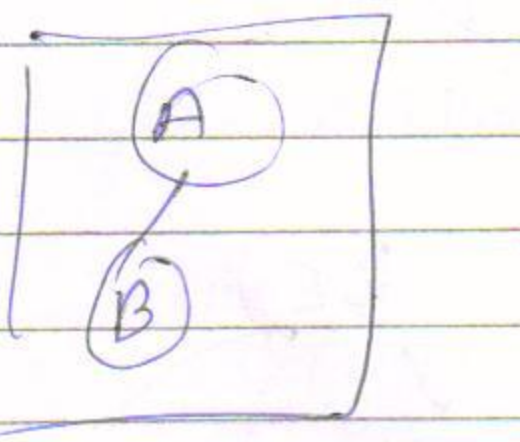
* disconnected graph

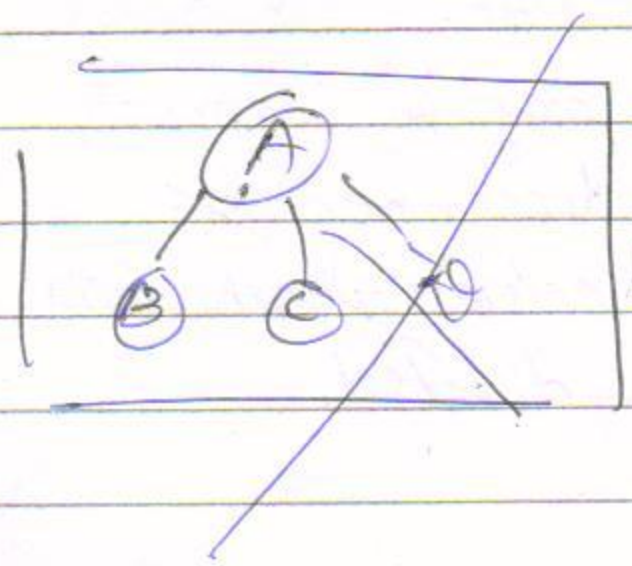


Binary tree :- Maximum 2 children

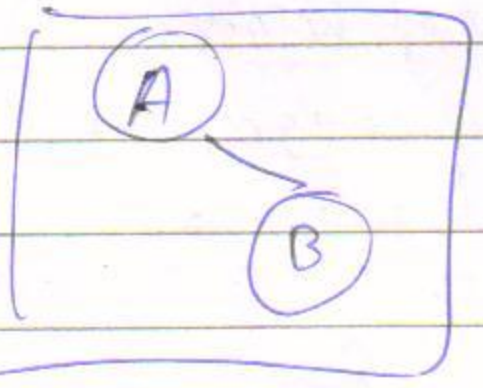


degree ⇒ no of children



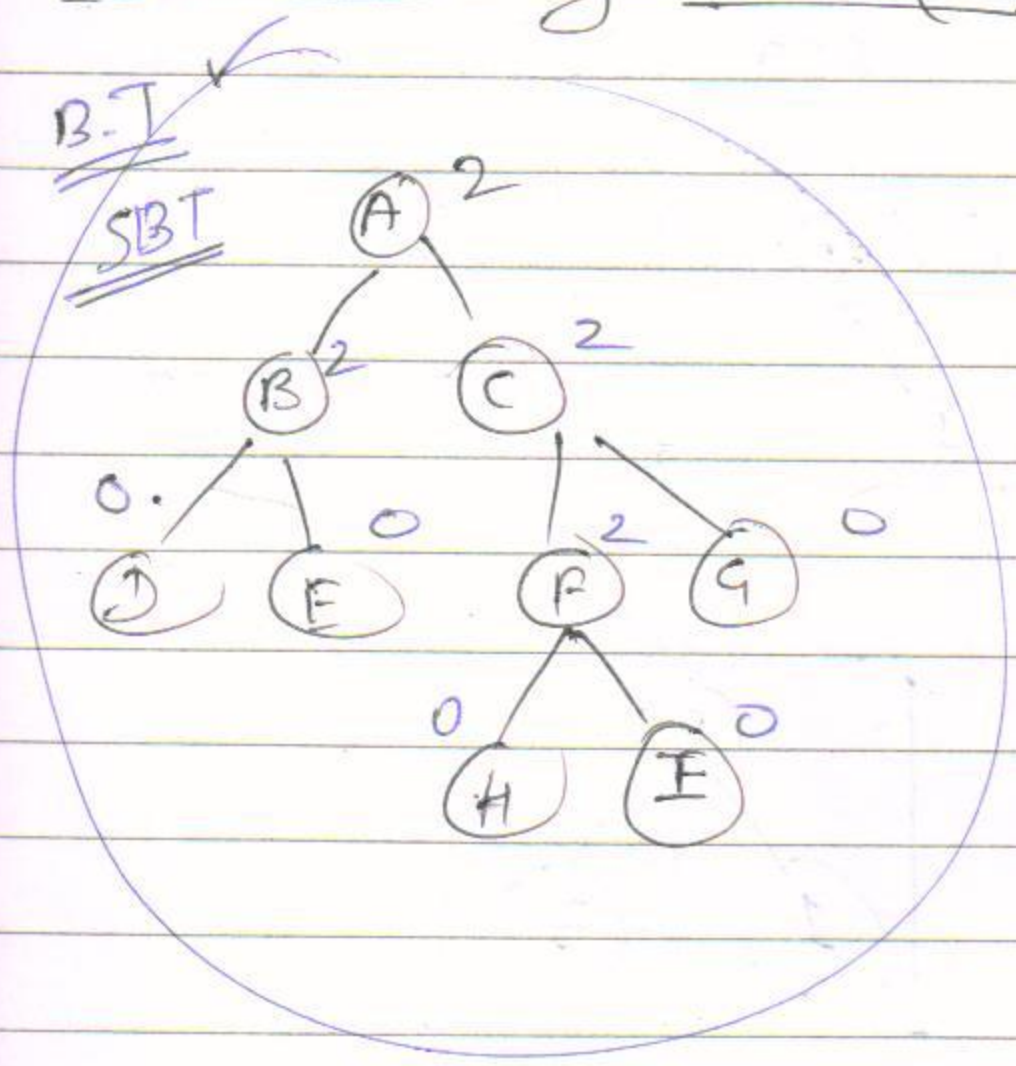


NO B.T.



Date

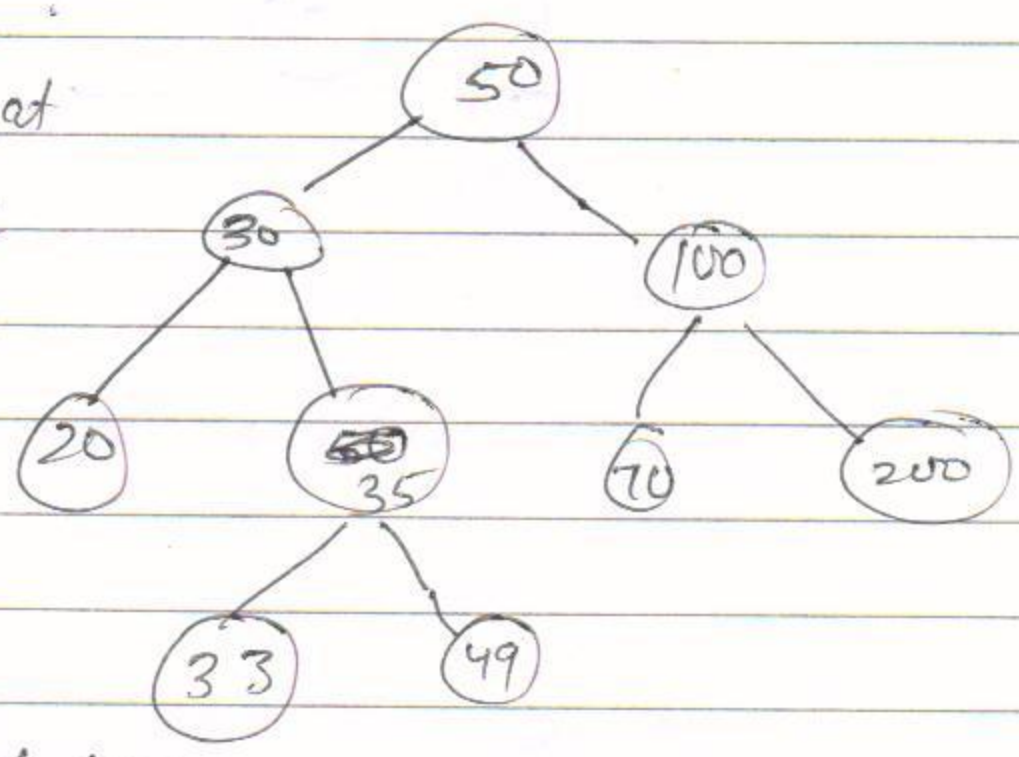
⇒ Strict Binary Tree (Full Binary Tree) :-



Binary search tree :- equal elements can be placed either left or right.

⇒ Definition :-

- In the given binary tree at every node root is greater than or equal to all elements presents in the Left subtree, & Root is less than or equal to all element present in the Right subtree, then it is called Binary search tree.



Q. How many Binary search tree posible with n distinct nodes :—

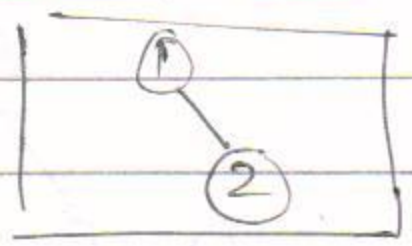
Ans.

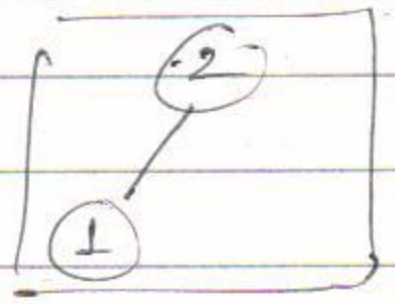
Date

$n \Rightarrow 0$ $\Rightarrow$ 1 BST

$n = 1$ $\Rightarrow$ 1 BST

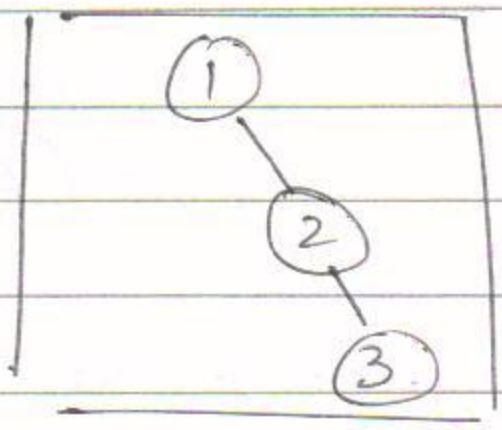
$n = 2$ (1, 2)

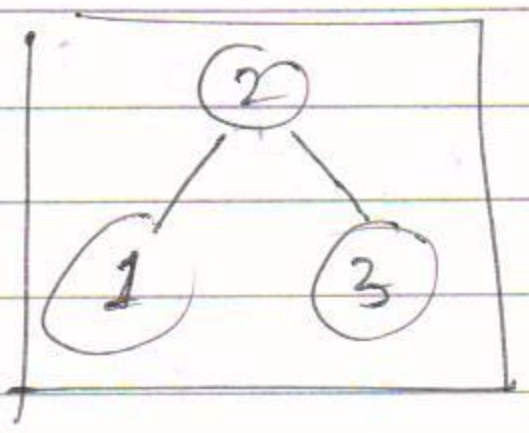




$\Rightarrow$ 2 B.S.T.

$n = 3$ (1, 2, 3)

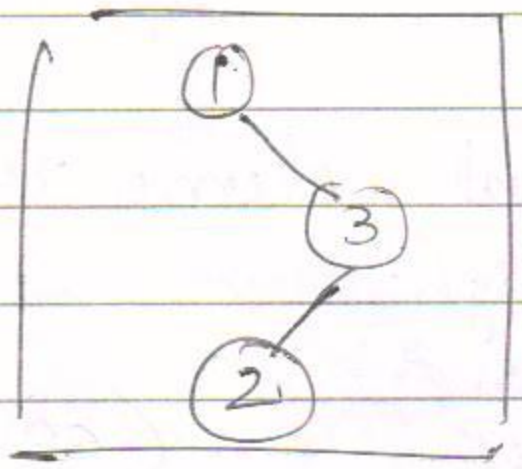


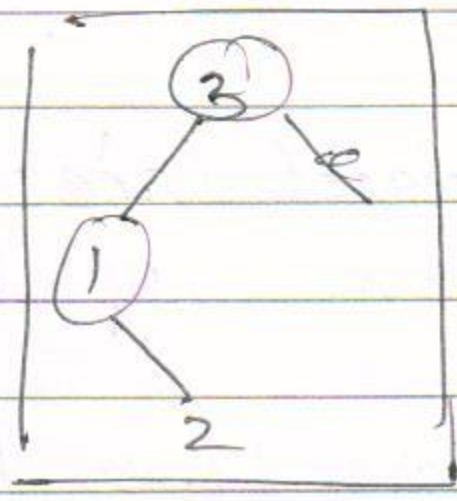


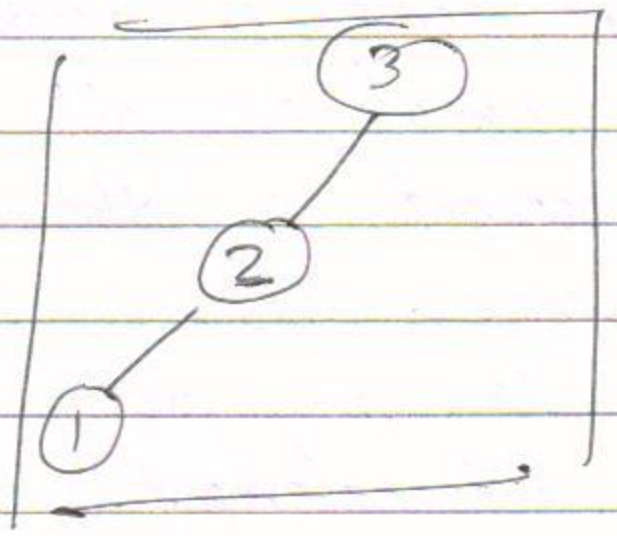
(1)

1 2

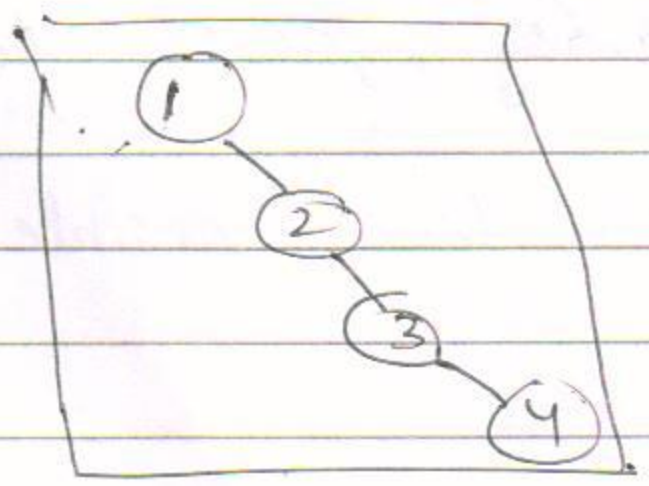
$1 \times 2 = 2$

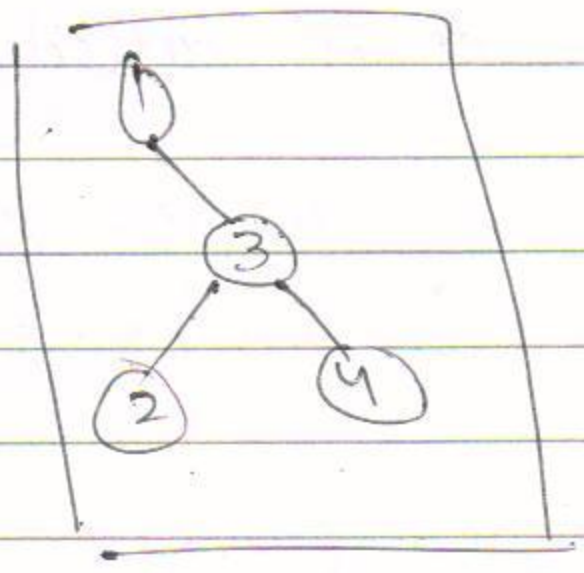




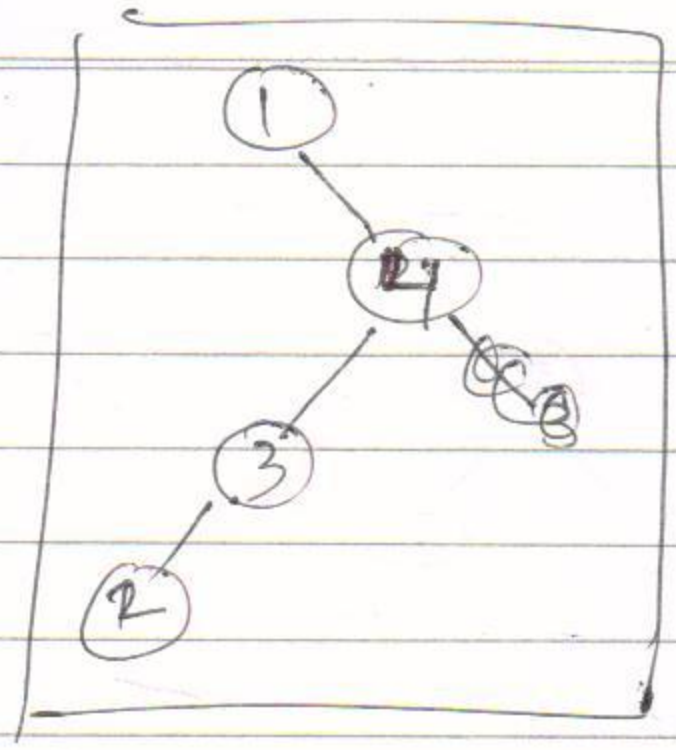


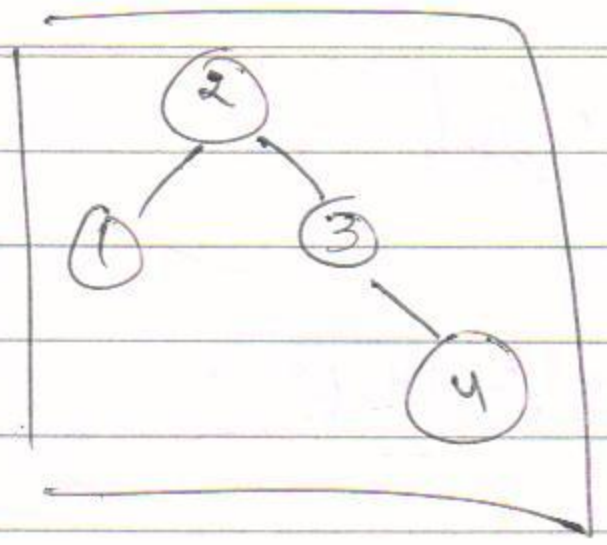
$n = 4$ (1, 2, 3, 4)

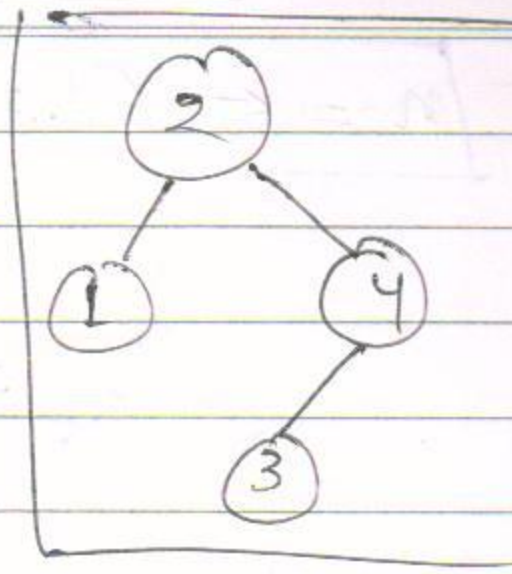


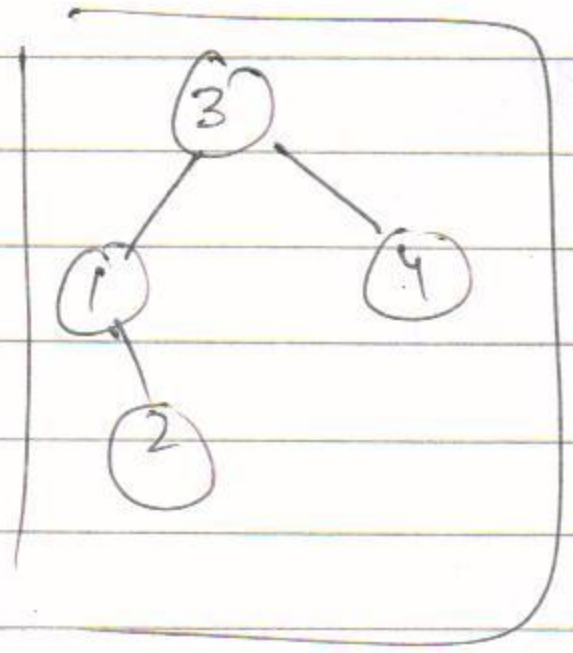


Date

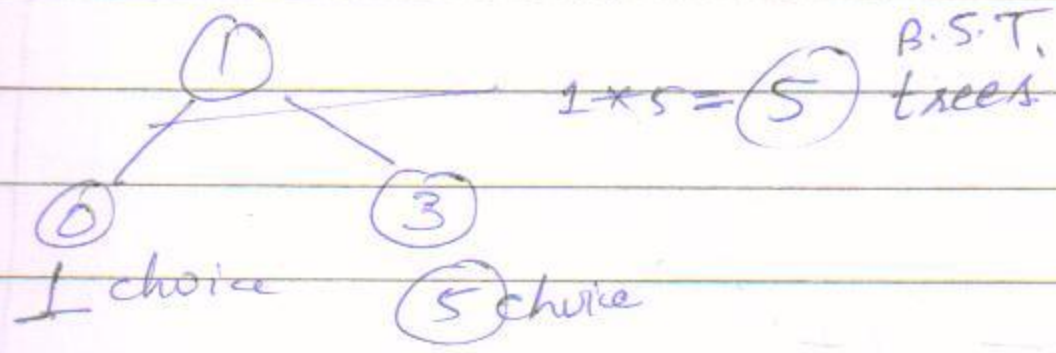


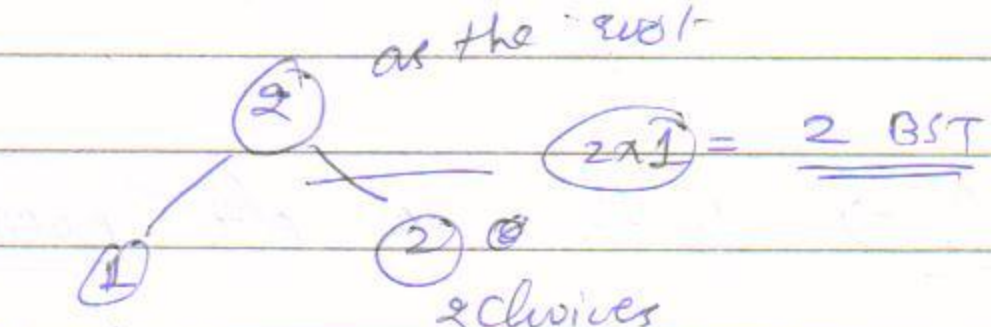


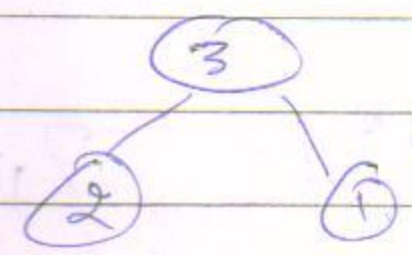


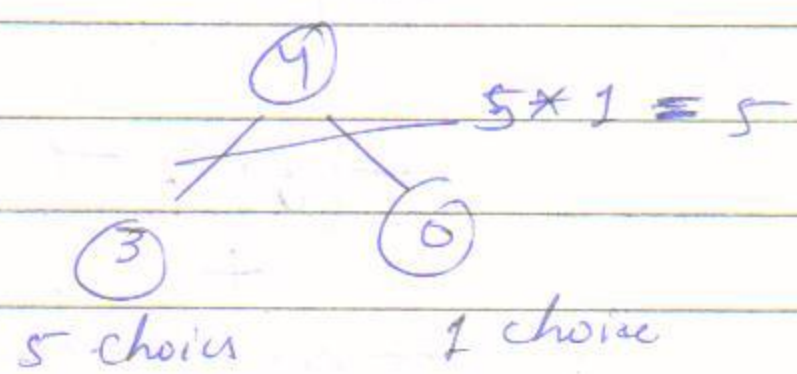


$n = 4$ (1, 2, 3, 4)







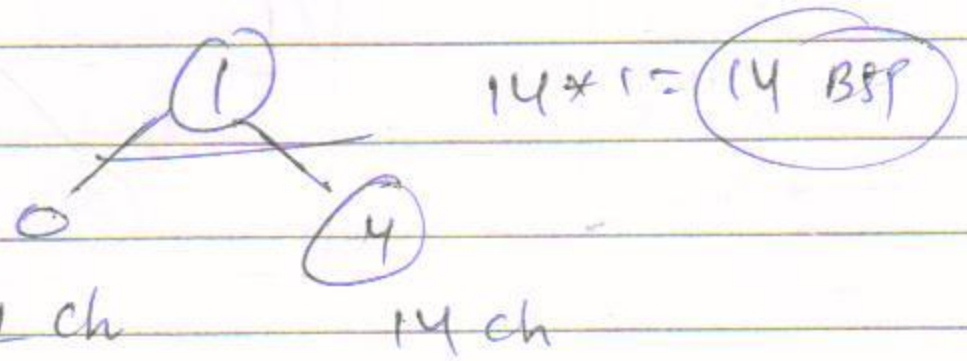


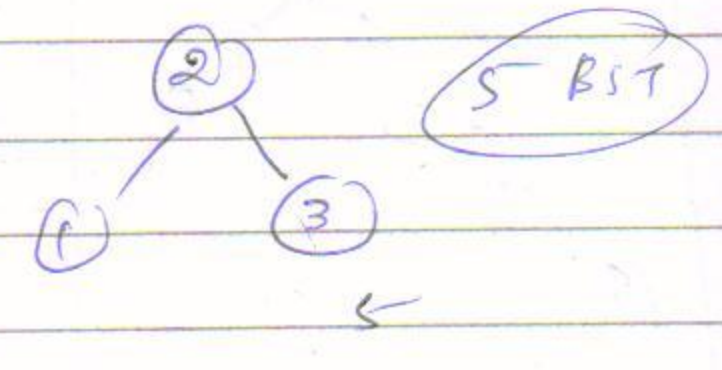
$5 + 2 + 2 + 5 = 14$ B.S.T.

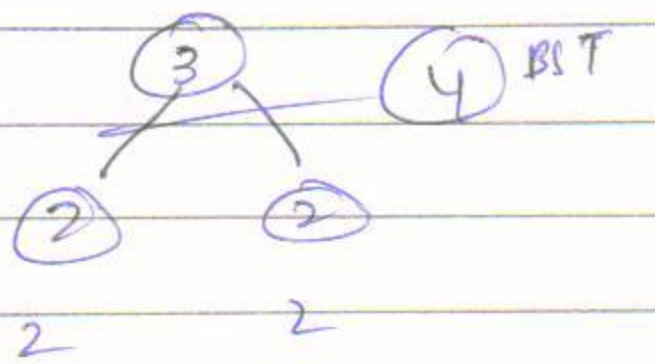
Q. How

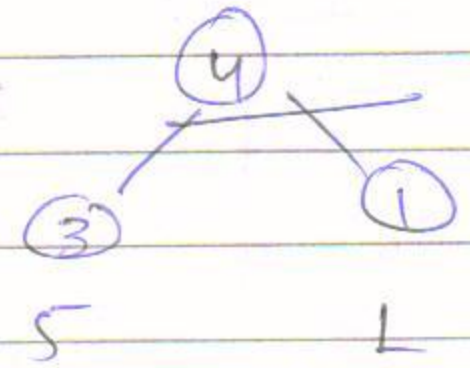
Date

$n = 5$ (1, 2, 3, 4, 5).

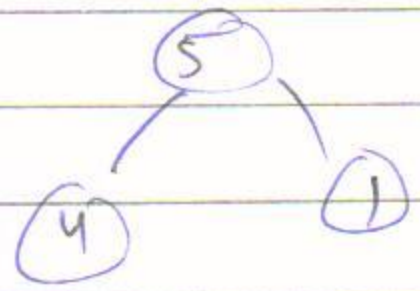








(14) BST



Total BTS.T. = 14 + 14 + 5 + 5 + 4

= 28 + 10 + 4

= 42

Q. ⇒ With 5 nodes

$$T(n) = \sum_{i=0}^{n} T(i) \cdot T(n-(i+1))$$

If

$T(5) = T(0) \cdot T(4)$

+

$= T(1) \cdot T(3)$

+

$= T(2) \cdot T(2)$ +

$= T(3) \cdot T(1)$ +

$= T(4) \cdot T(0)$

⇒

$= 1 * 14 +$

$1 * 5 +$

$2 * 2 +$

$5 * 1 +$

$14 * 1$

⇒

14
+
5
+
4
+
5
+
14

42

Date

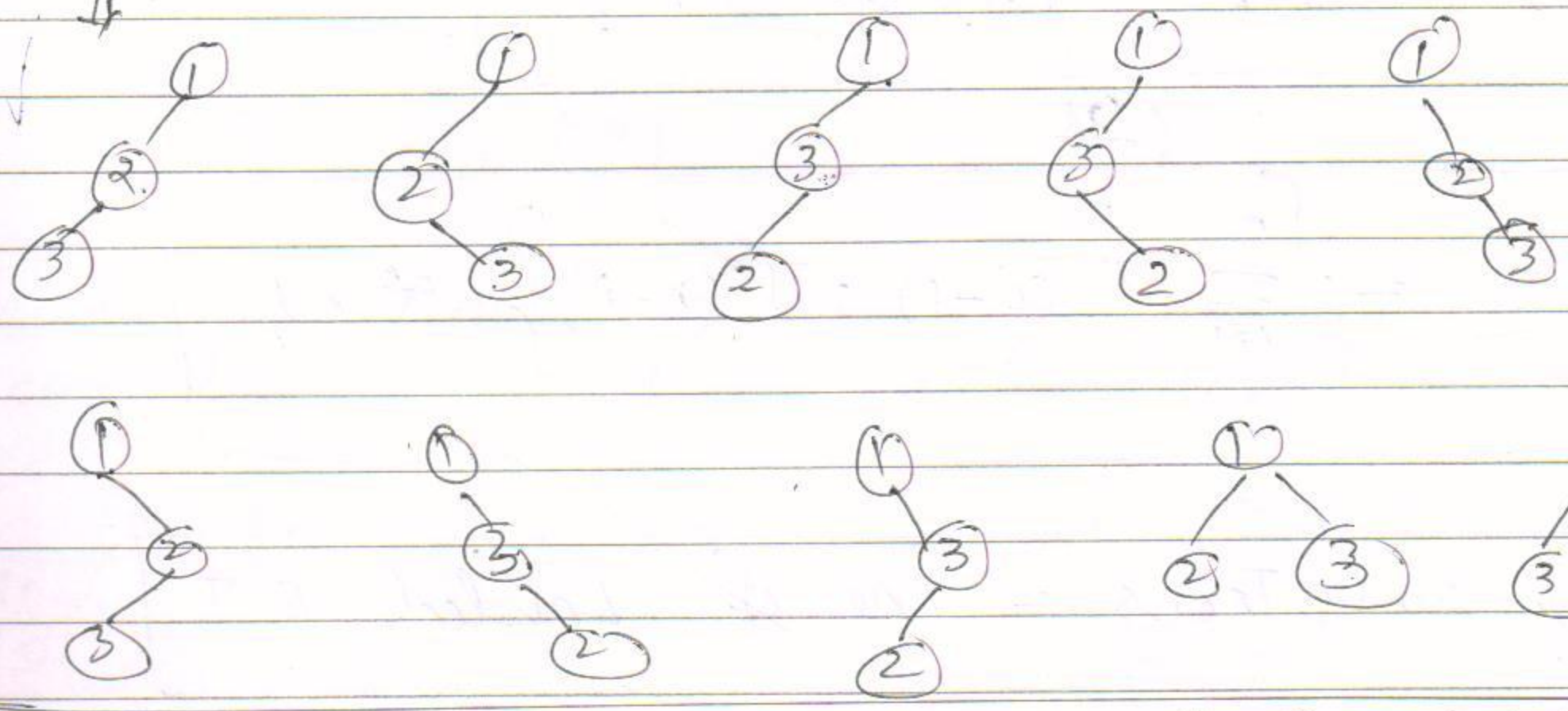
Let $T(n)$ be the total no of Binary search Trees posible with n element then

$$T(n) = \sum_{i=1}^{n} T(i-1) \cdot T(n-i)$$

0	4	⇒
1	3	⇒
2	2	⇒
3	1	⇒
4	0	⇒

Q. How many Binary trees posible with n nodes.

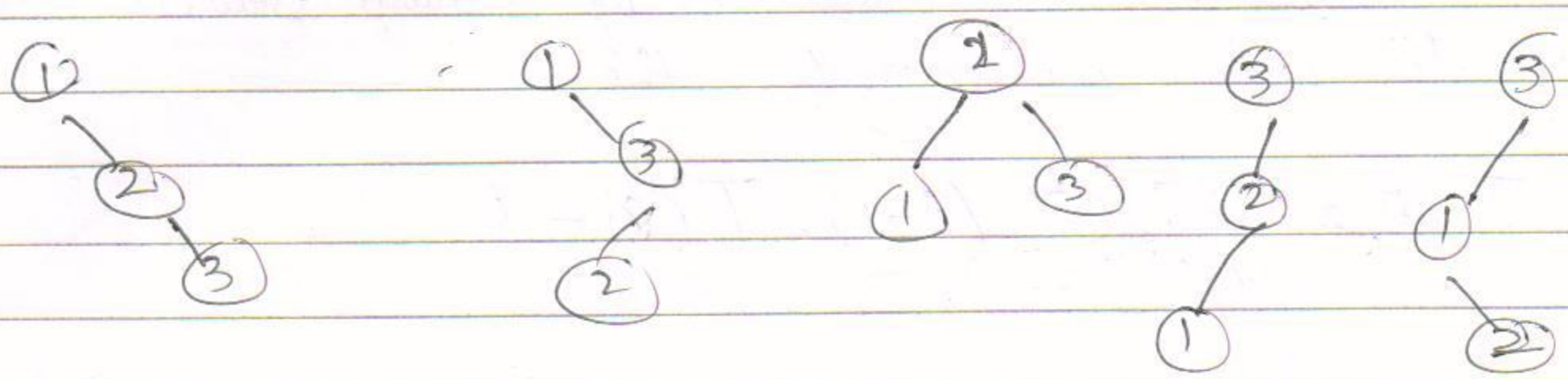
$n=3$ (1, 2, 3)



with (1) root ⇒ 10
with (2) root ⇒ 10
(3) root ⇒ 10
total, 30

unlabled Binary tree = BST.

n=3 (1, 2, 3) :- In Binary search tree



n=4 : with 4 nodes # of BT ?!

BST with (4) * 4!

= 14 * 24

= 336.

$$NBT(n) = NBST(n) * n!$$

(OR)

$$NBT(n) = \left\{ \sum_{i=1}^{n} T(i-1) \cdot T(n-i) \right\} * n!$$

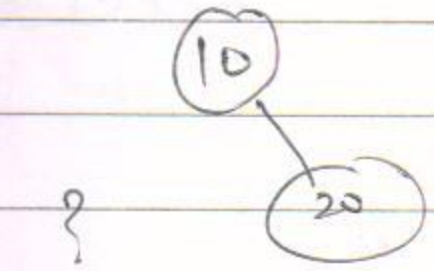
No. of Binary Trees = no. of Labled B-T.

No of BST = No of unlabled B-T

3-Nov-12

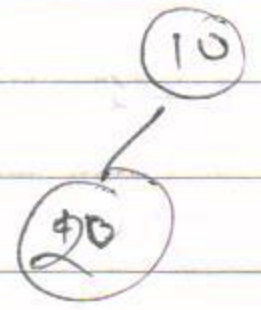
Date

⇒ Almost Complete Binary Tree :-



BT ✓

Not ACBT

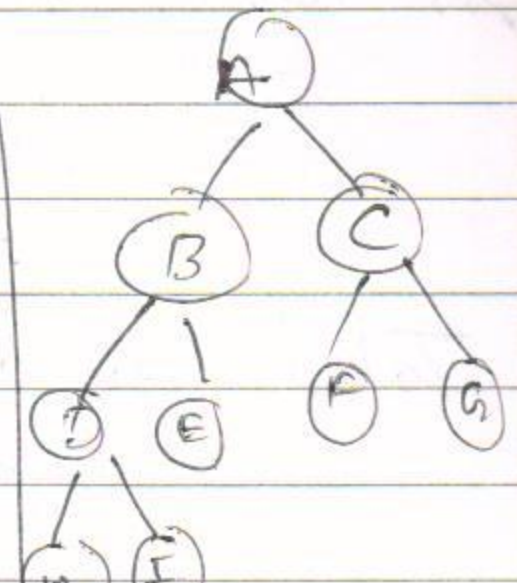


BT

ACBT ✓

BT ✓

ACBT ✗

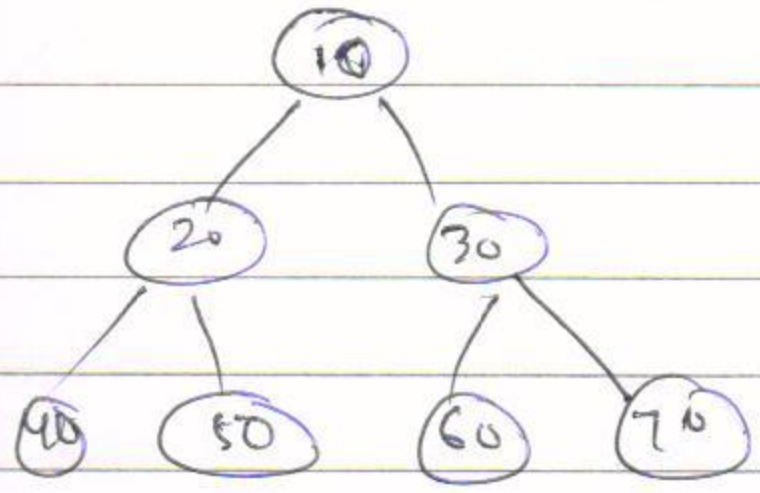


BT ✓

ACBT ✓

⇒ Definition:-

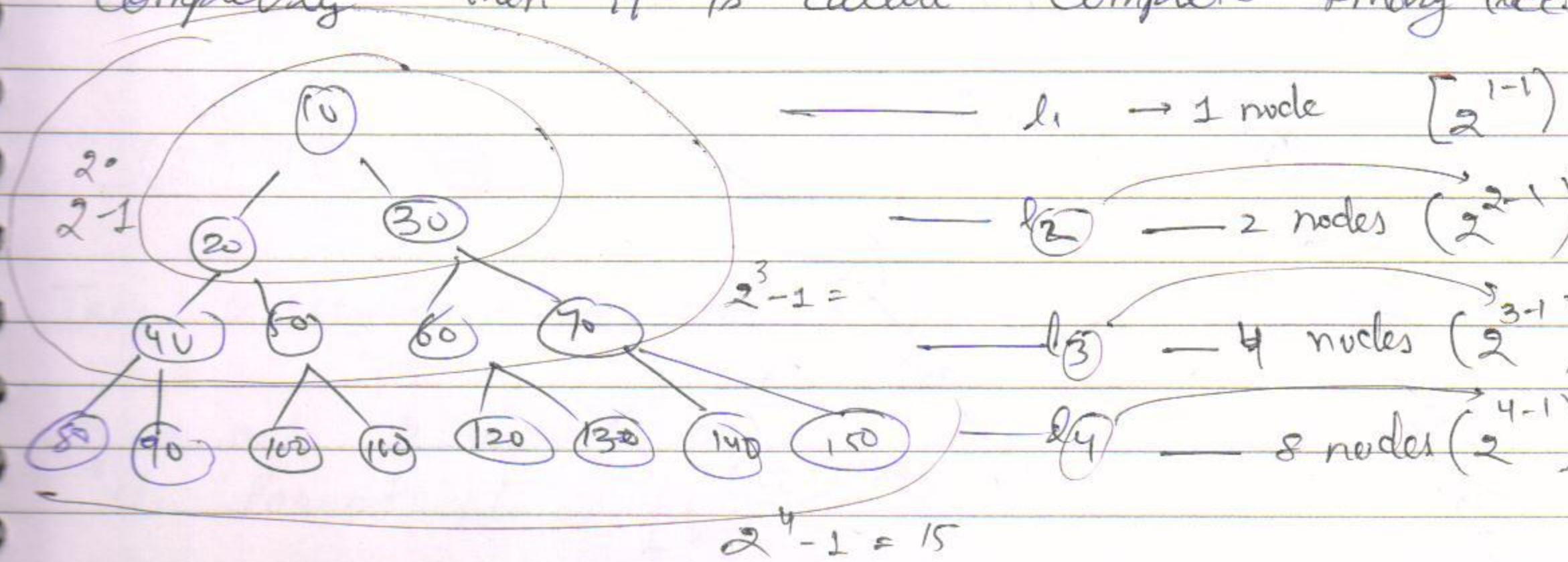
In the given Binary tree at every node after completing left only go to right and after completing the current level only go the next level then it is called Almost Complete B.T. (ACBT.)



ACBT ✓

CBT.

In the given ACBT if last level also filled completely then it is called Complete Binary tree.



Maximal Binary tree.

Date

So No. of nodes at level i of Complet Binary tree or Maximal B.T. $= 2^{i-1}$

Total No. of nodes in a k level complete binary tree $= \left(2^{k} - 1\right)$

Relation:- b/w k & n

n = Total no of nodes

k = no. of levels.

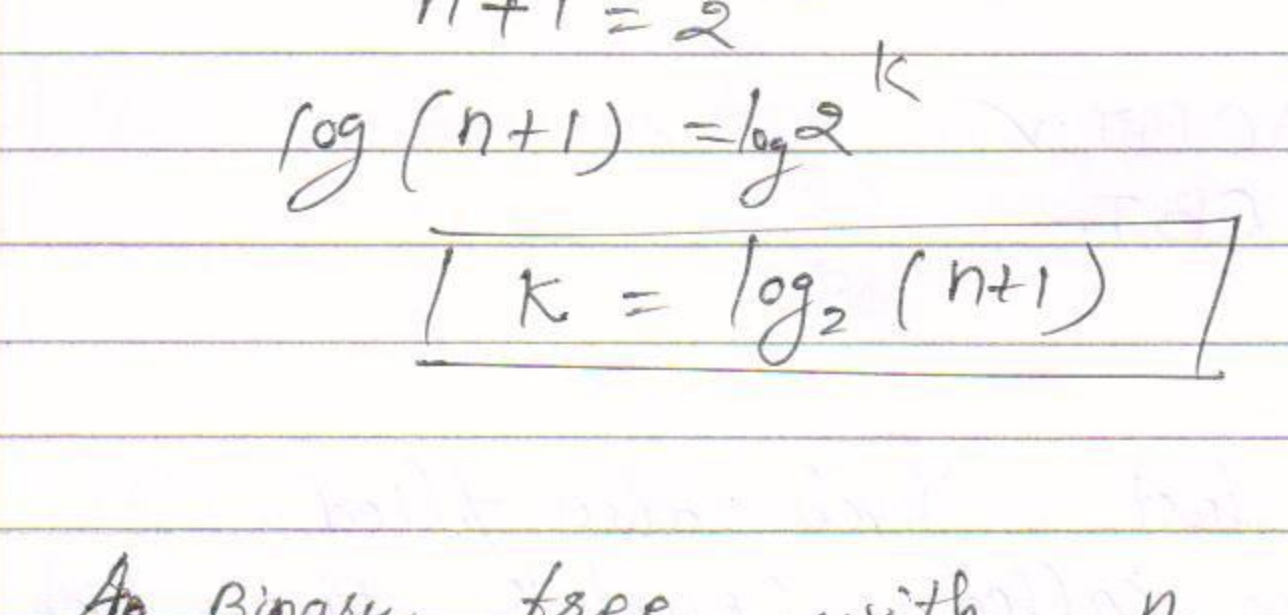
$$n = 2^{k} - 1$$

$$n+1 = 2^{k}$$

$$\log(n+1) = \log_2 2^{k}$$

$$\boxed{k = \log_2(n+1)}$$

A Binary tree with n



Date

B.T	B.T	B.ST	ACB.T (or)
if n nodes then	+	+	CB.T
(max) levles. →	n	n	$\log_2(n+1)$
levels (min)	$\log_2(n+1)$	$\log_2(n+1)$	$\log_2(n+1)$

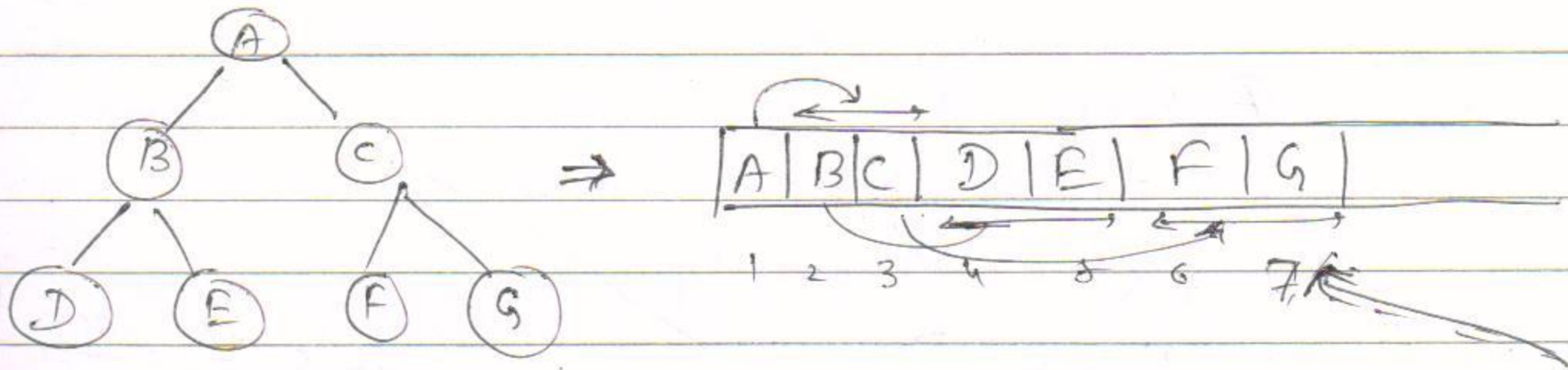
⇒ How Binary Tree is stored in computer ?

→ greatest drawback with array

10	20	30	40	50	60	

if we want to insert at the 1st place: all the element should be right shifted.

⇒ In online tranjactions most probably we use link list. But drawback of link list is No random access.



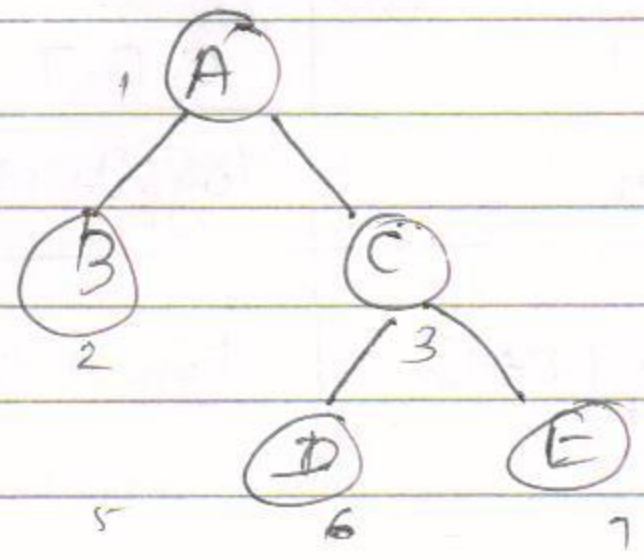
Tree is stored in the computer in the form of array

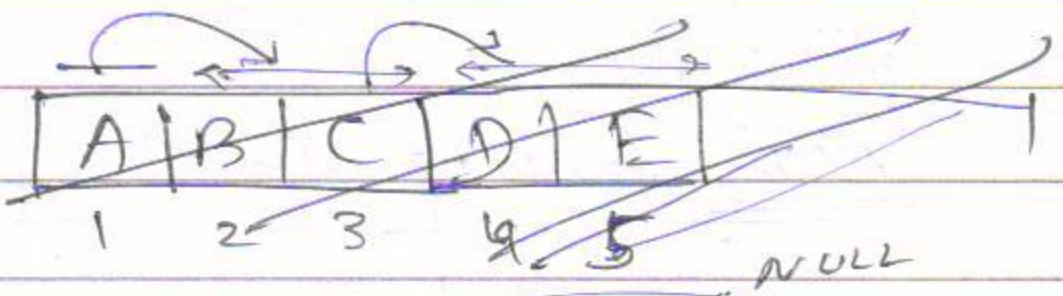
If a node is stored in i^{th} place of array -

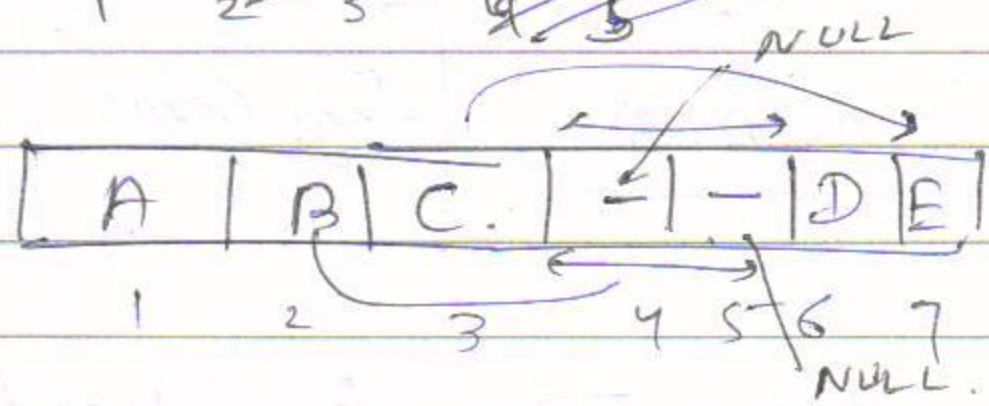
(i) Parent $(i) = \left\lfloor \frac{i}{2} \right\rfloor$ —— floor value

(ii) Left child of $i = 2 * i$

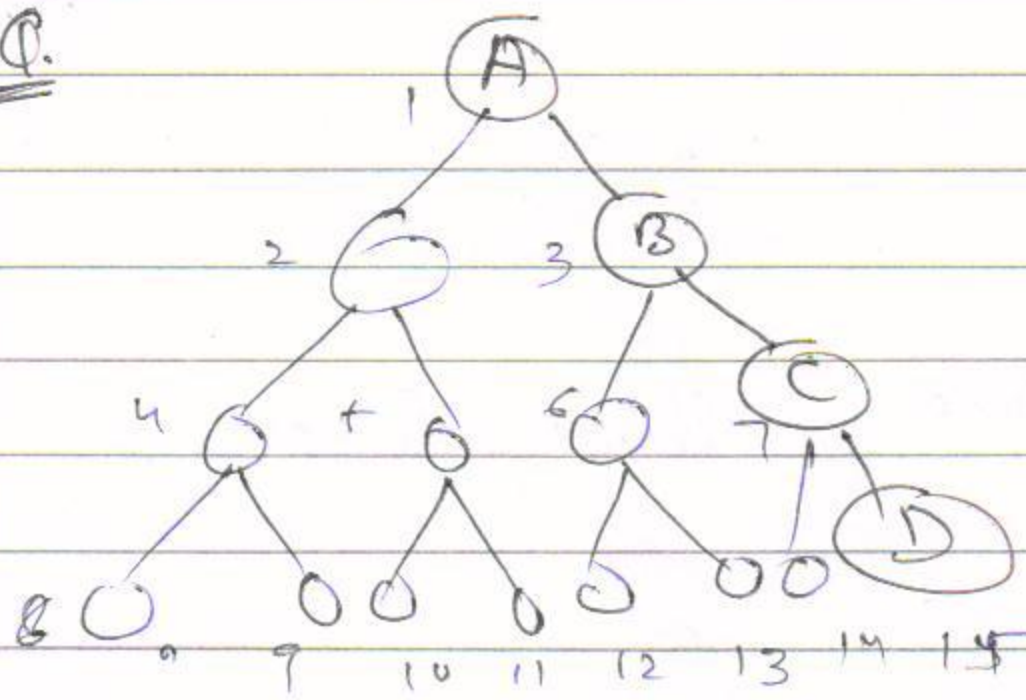
Q. Store the following tree in the form of Date array.

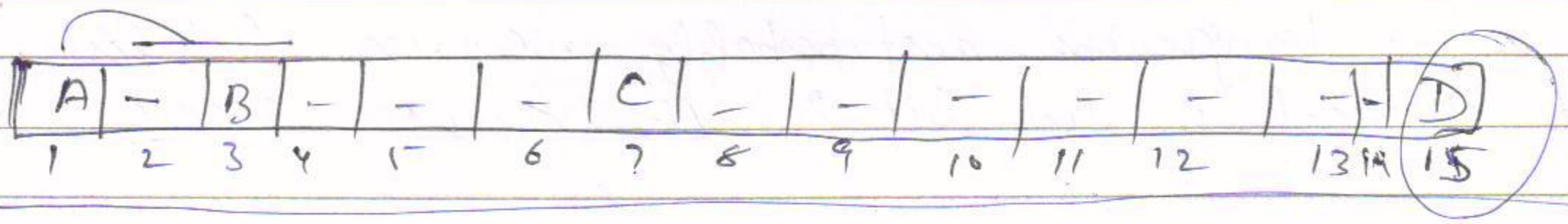




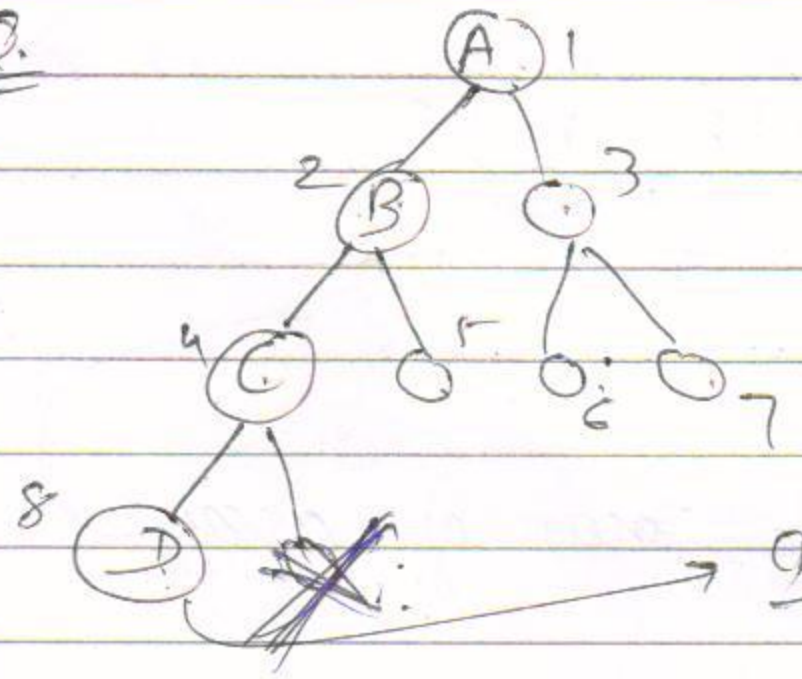


Q.

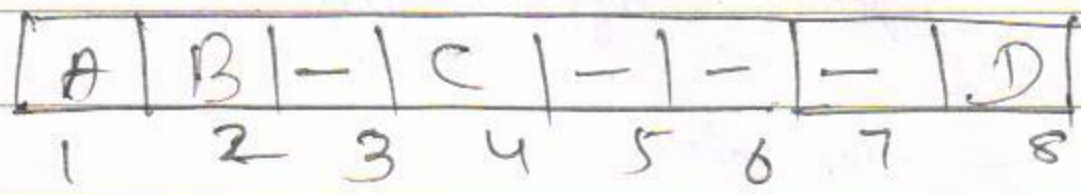




Q.



→ as soon as last node is seen → stop



Q.. Binary tree contains n-nodes then how many array slots

B. Tree

max size of array ⇒ $2^n - 1$ → (there may be gap)

minimum slots of array ⇒ n (in case of ACBT or CB.T.

Note :-
Binary tree represesentation using array is better idea if Binary tree is ACBT. or CBT.

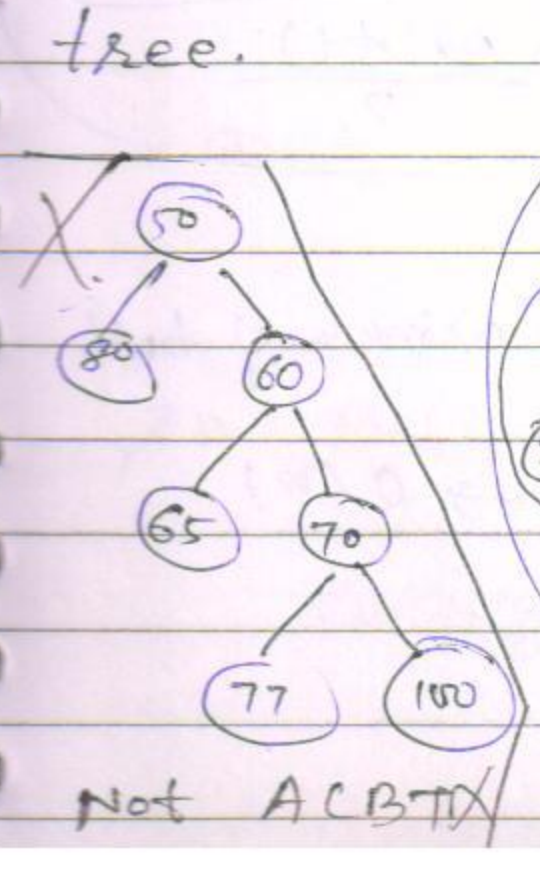
⇒ Heap Sort :-

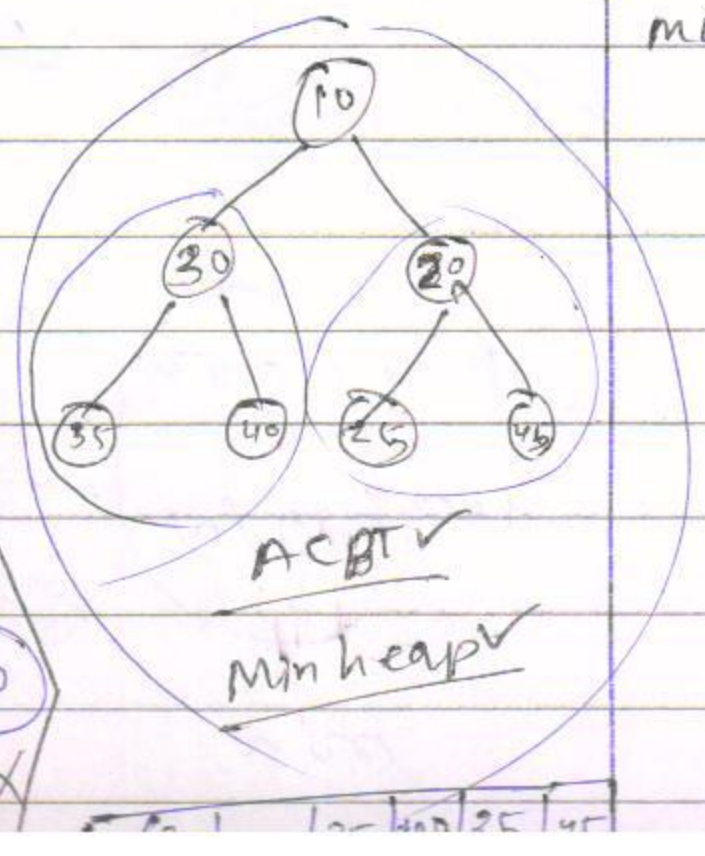
Heap-Tree :- Heap trees are of 2 types.

Heap Tree

* Min heap Tree

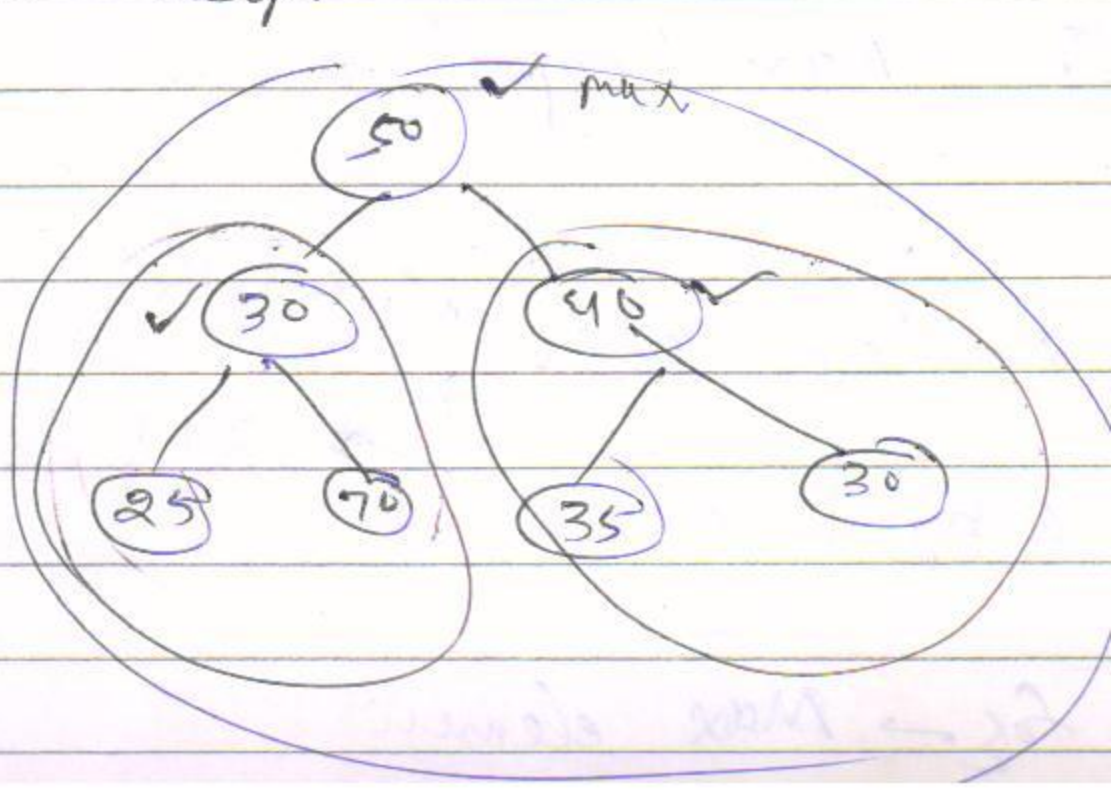
Defination :-
In the given ACBT, at every node root is minimum or equal comparing it's children then it is called min-heap tree.





* Max heap tree.

In the given ACBT, at every node root is maximum or equal to comparing it is children then it is called max-heap.



Date

if in minheap

take O(1)

Min heap —— O(1)
Array —— O(n)

min

O(n²)
O(n) —— O(n)

max

Note- In minheap tree to find maximum element also we take O(n) Becaus if also stored in array.

Max heap:-

Comparisons.

1st max → 0
2nd max → 1
3rd max → 2 compari
4th max → 3 Com
5th — 4
:
:
1000 99

TC = [O(1)]

Sum of 99 Natural no.s.

n → n(n+1)/2

In Max heap.

Max heap O(1)
array O(n)

min of both → O(1).

fix → Max element

Max O(n²)
array O(n) → O(n)

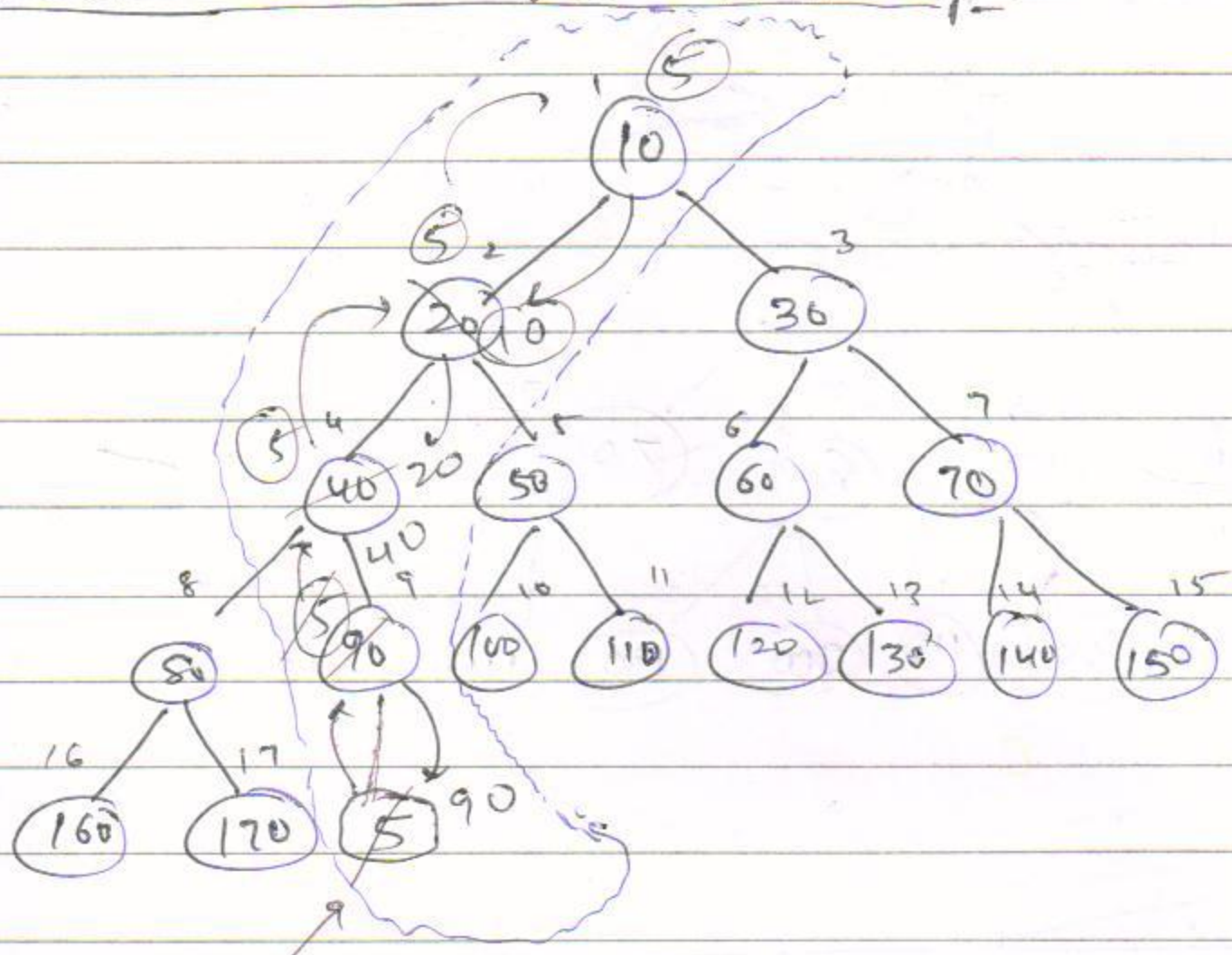
minimum of both

min

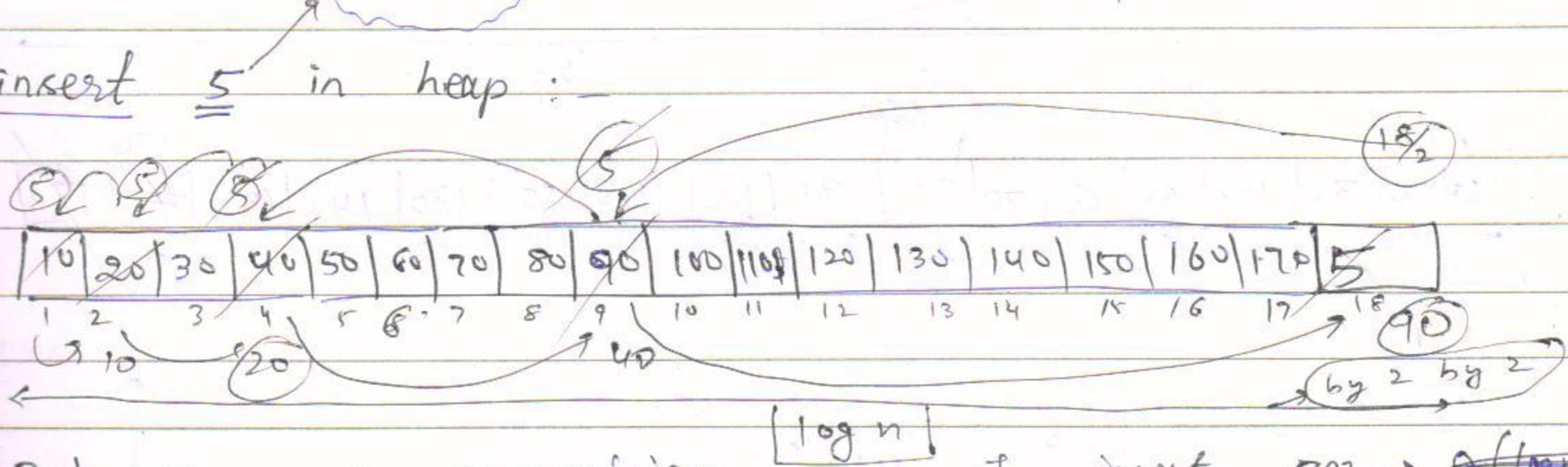
Date

⇒ Min heap :-

& Insertion into the min heap :- using array representation



→ insert 5 in heap :-



Best case → 1 compasision if insert 500 ⇒ ~~$\Theta(\log$~~

Worst case → log n com if inset 5 ⇒

average case →

$$\text{Total} = 1 + 2 + 3 + 4 + 5 + 6 - - - - - + \log n$$

$$= \frac{\log.(\log n+1)}{2} = (\log n)^2$$

$$\text{average} = \frac{(\log. n)^2}{\log n} = \theta(\log n)$$

so Best case ⇒ $\Omega(1)$

Worst case ⇒ $O(\log n)$

average Case ⇒ $\Theta(\log n$

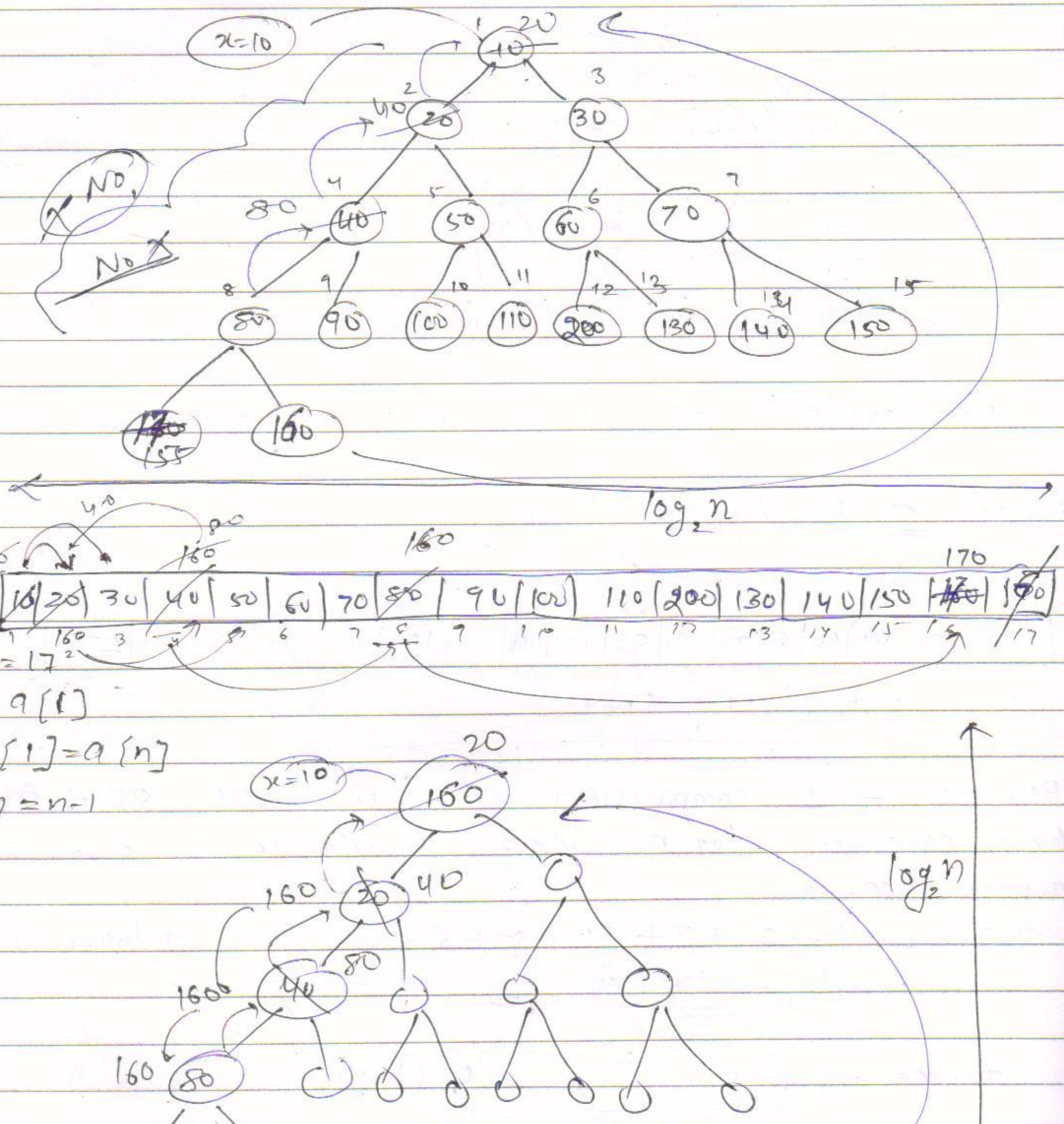
Q. Show the result after deleting 2 times.

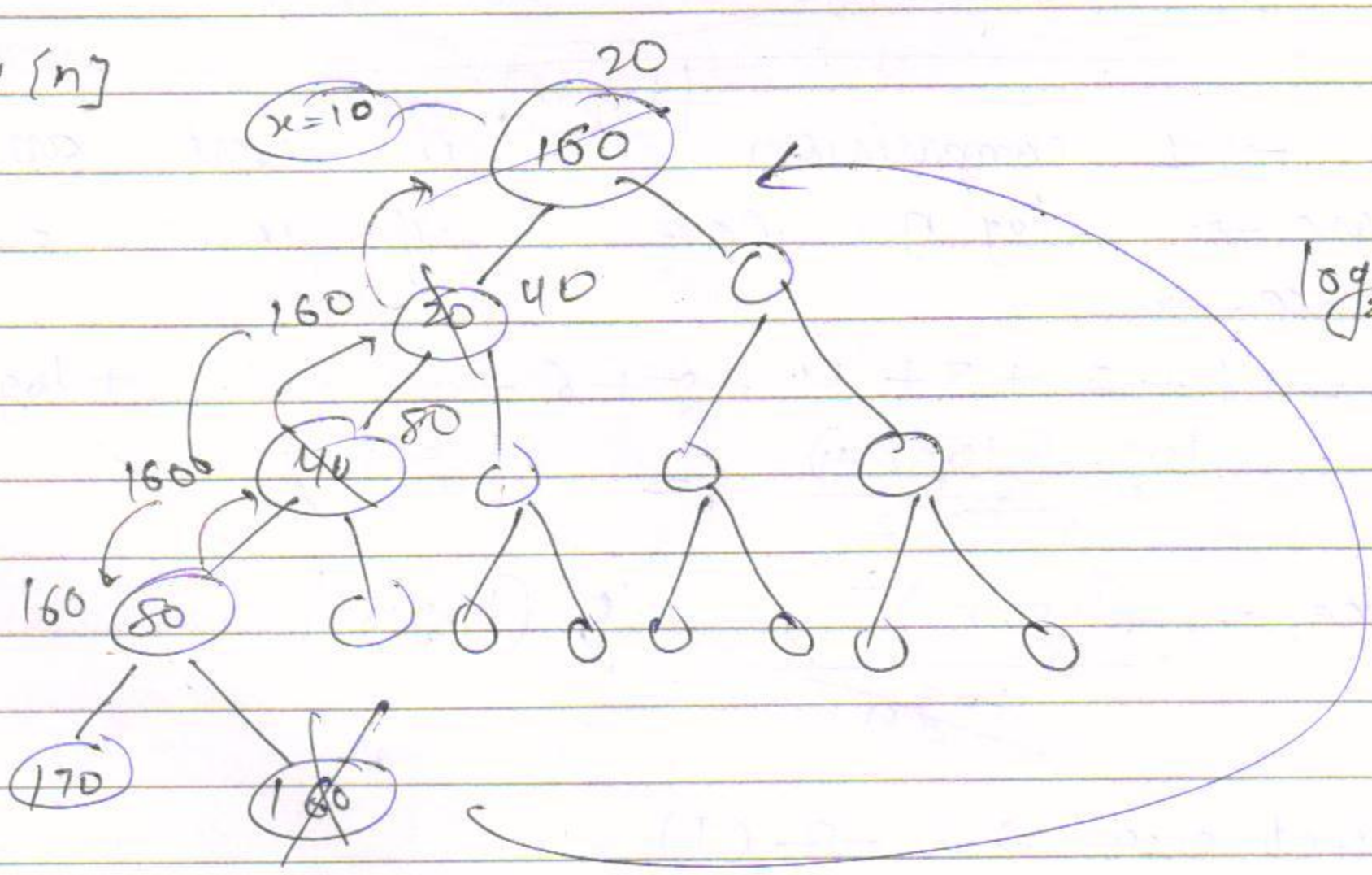
180

Date

⇒ Min heap:-

→ Deletion in Min heap:-

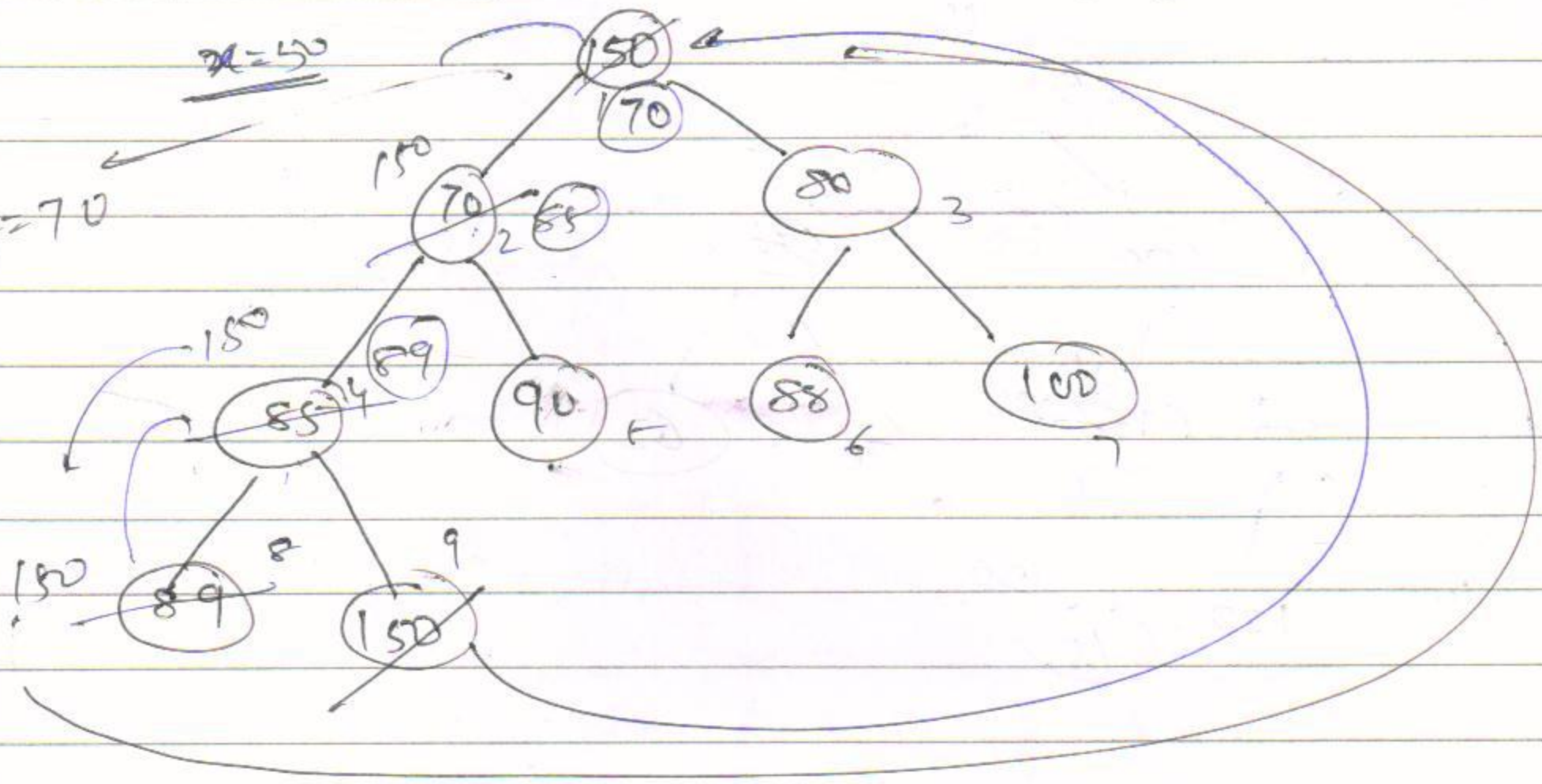




Procedure :- ① x=10
② a[i] = a[n]
③ n = n-1

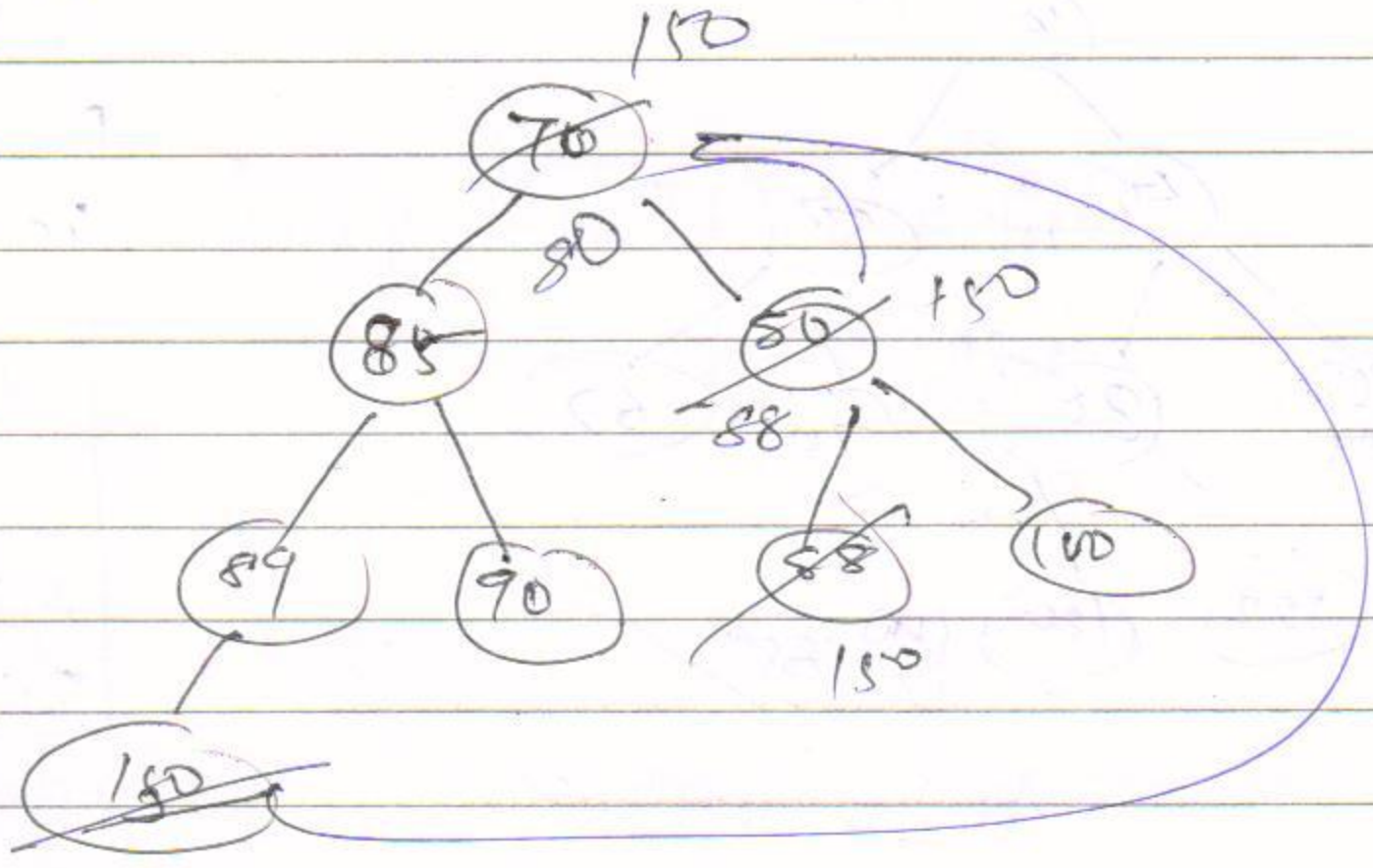
Best case ⇒ Ω(1)
Worst ⇒ O(log n)
Average Case ⇒ O(log n)

Q. Consider the following min-heap tree. Show the result after one deletion.



70	85	80	89	90	88	100	150		
1	2	3	4	5	6	7	8	9	

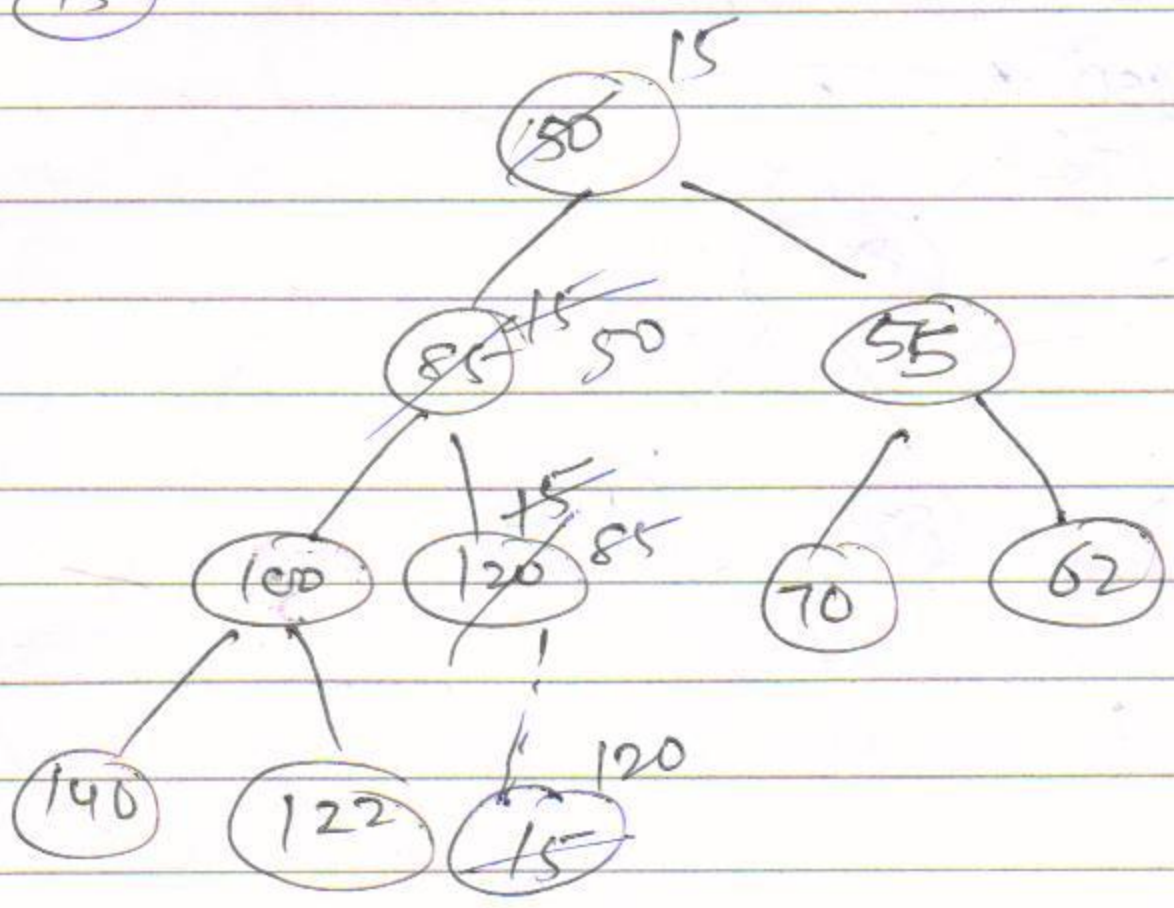
one more time



80	85	88	89	90	~~88~~ 150	100

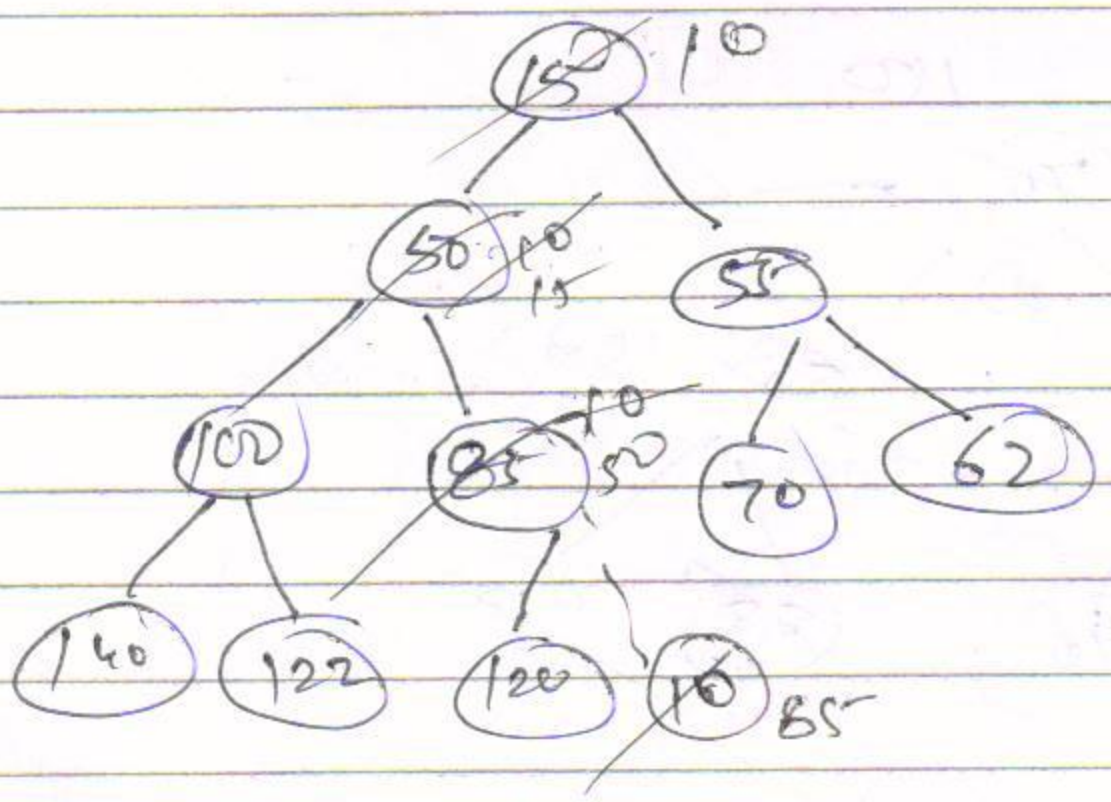
Q.. Consider the following min heap — Date

insert (15)



15	50	55	100	85	70	62	140	122	120

One more insertion → I — 10



like inserting n element one by one

using Brute force Algo Creating Min heap →

TC = O(n log n)

10	15	55	100	50	70	62	140	122	120	85

Note:- Creating min heap tree using Build heap Algo {O(n)} time

Date

```
Build_Heap (a, n)
{
    for ( i = ⌊n/2⌋ down to 1)
    {
        min_heapify (a, n, i)
    }
}
```

```
min_heapify (a, n, i)
{
    l = 2i ;    r = l + 1 ;
    min = i ;
    If ( l ≤ n  && a[min] > a[l])
        min = l ;
    if ( r ≤ n  && a[min] > a[r])
        min = r ;
    if ( i ≠ min)
    {
        swap (a[i], a[min])
        Min_heapify (a, n, min) ;
    }
}
```

O(log₂ n)

Time complexity of MinHeapify procedure = O(log₂ n)

n = 31

m = ⌈n/2⌉ = 16

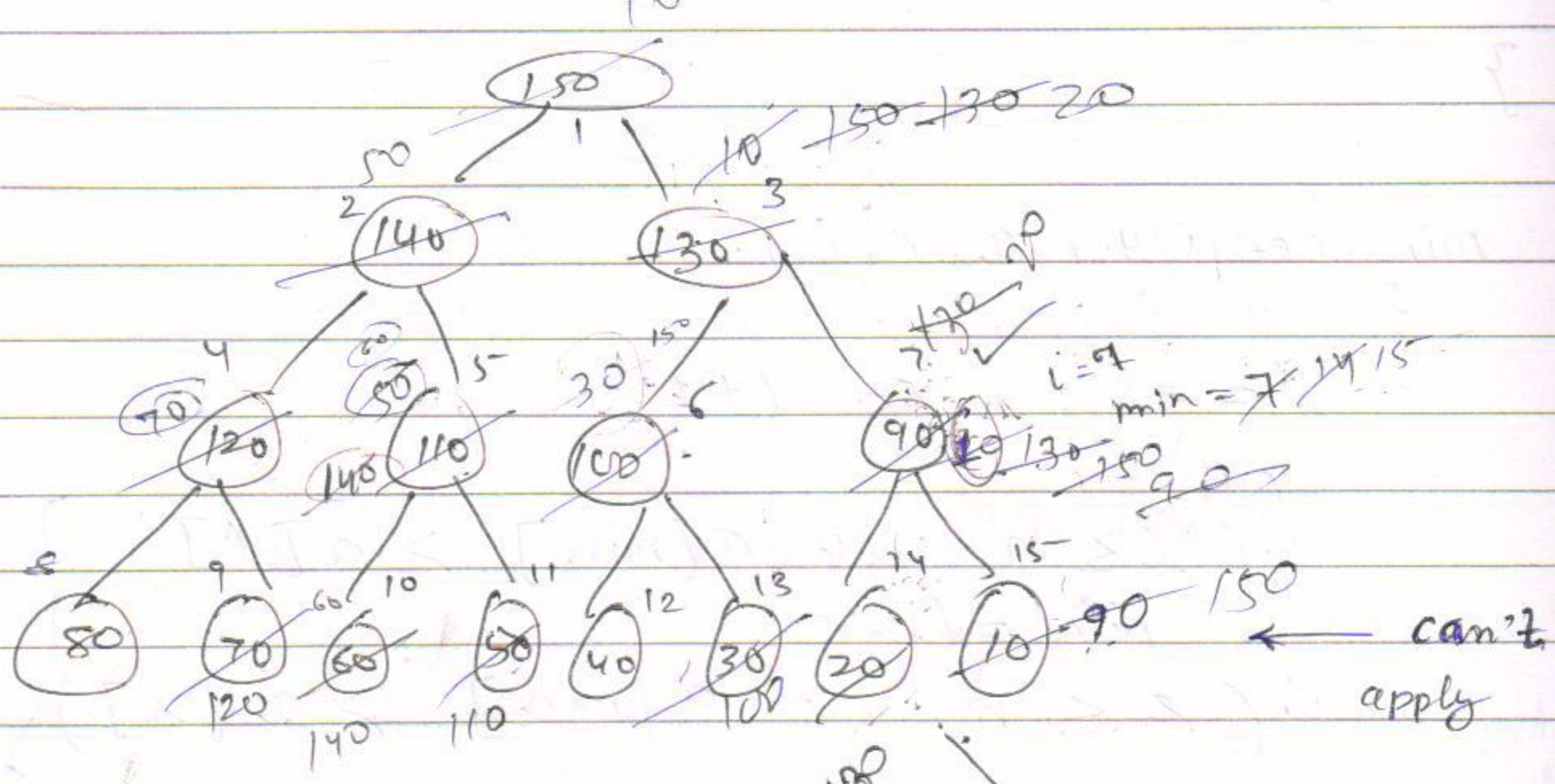
⇒ Construct min heap tree using following element, using build heap Algo. —

Date

150, 140, 130, 120, 110, 100, 90, 80, 70, 60, 50, 40, 30, 20, 10

Steps

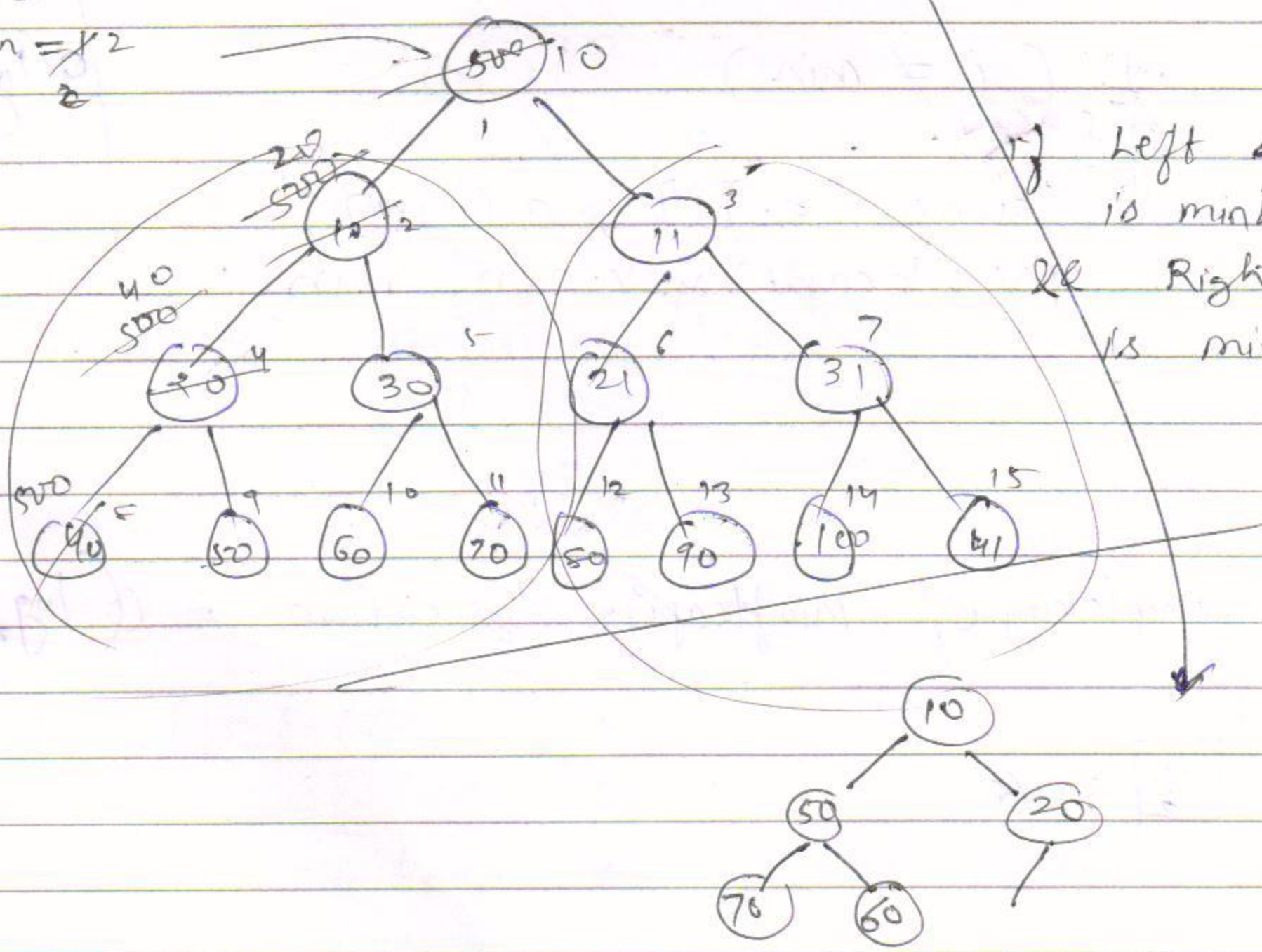
① create ACBT



i=1
min = 1 2

i=2
min = 2 4

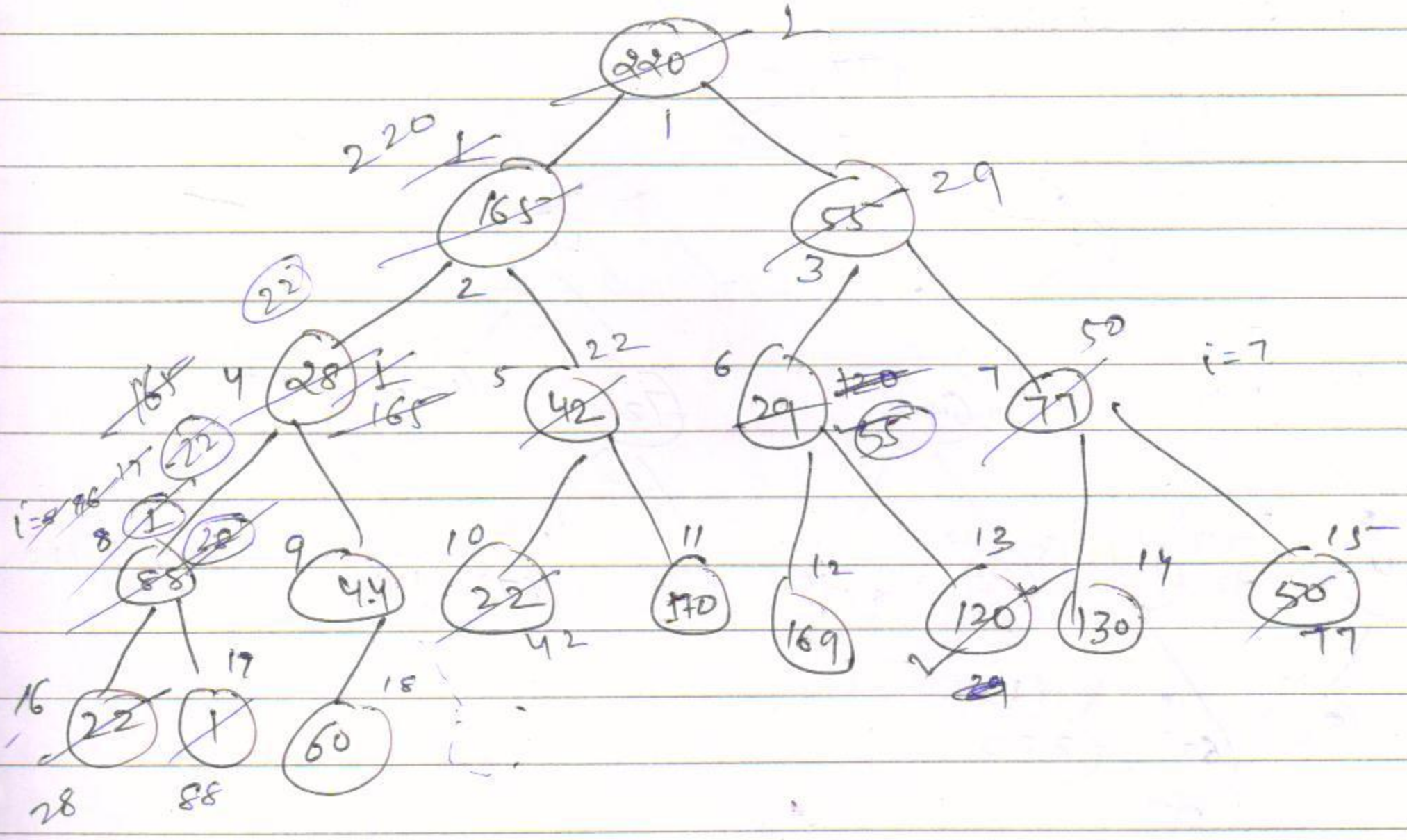
i=4
min = 4 8



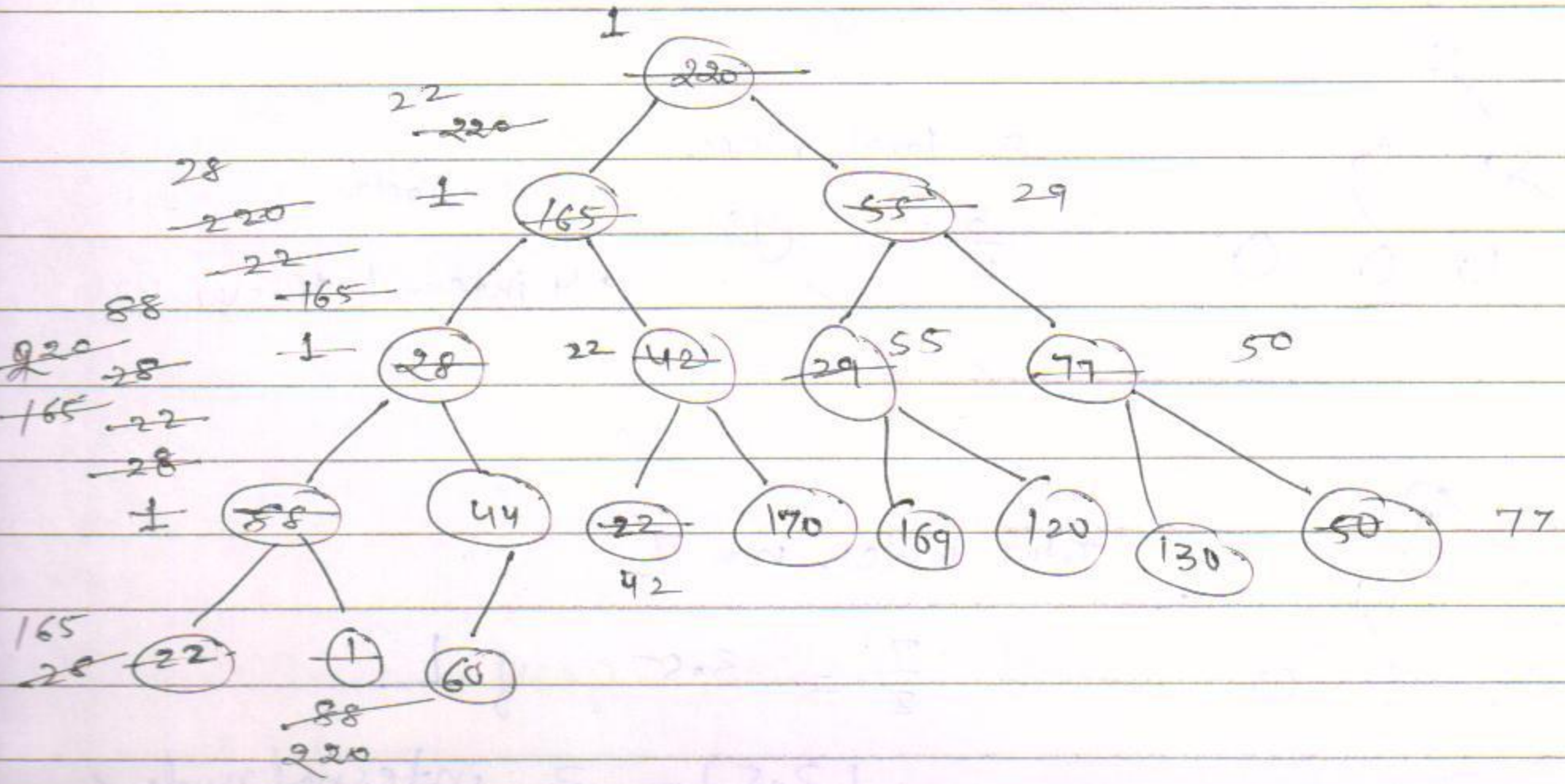
if Left subtree is minheap && Right subtree is minheap.

Construct min heap tree for following elements. Date

220, 165, 55, 28, 42, 29, 77, 88, 44, 22, 170, 169, 120
130, 50, 22, 1, 60,



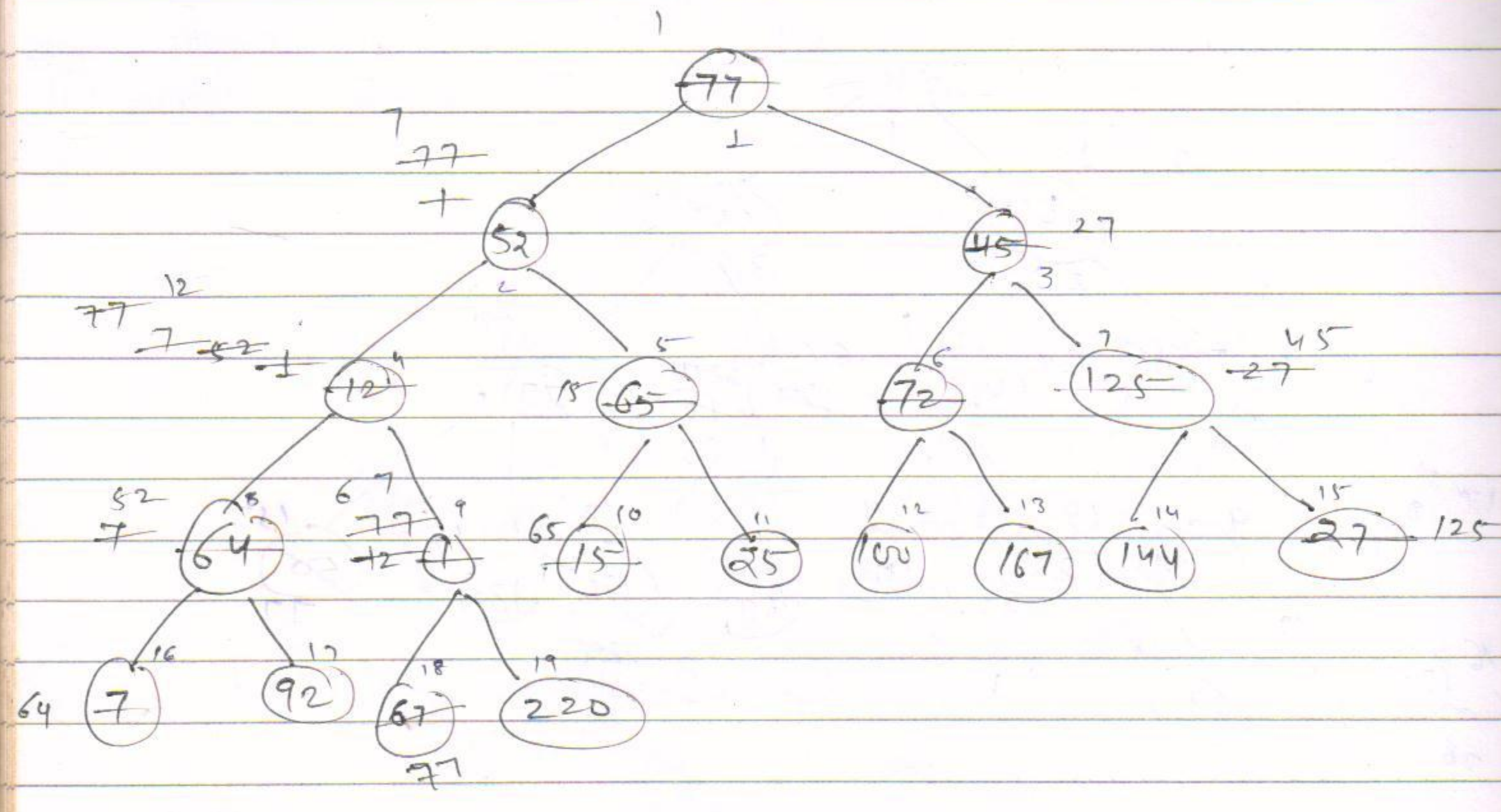
1, 22, 29, 28, 22, 55, 50, 88, 44, 42, 170,
169, 120, 130, 77, 165, 220, 60.



Date

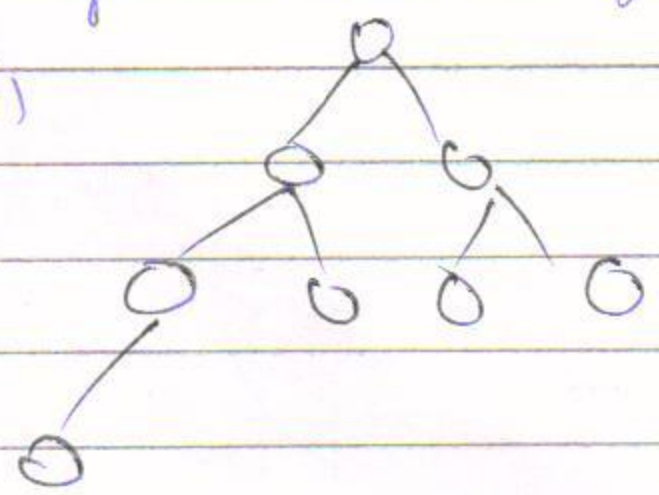
Q. Create Min heap.

77, 52, 45, 12, 65, 72, 125, 64, 1, 15, 25, 100, 167, 144, 27, 7, 92, 67, 220.



⇒ How to calculate # leaf nodes and no of internal nodes if Total no of nodes are given.

(i)

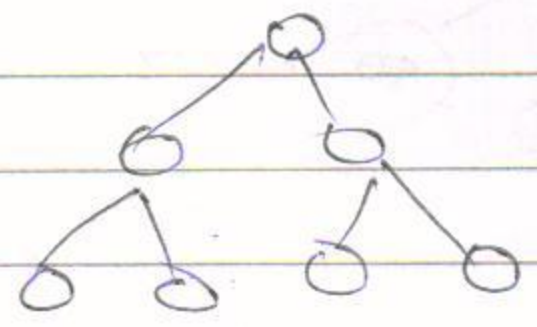


8 total nodes.

$\frac{8}{2} = 4$ leaf Nodes

4 internal Nodes

(ii)



Total nodes = n = 7

$\frac{7}{2} = 3.5$ if L

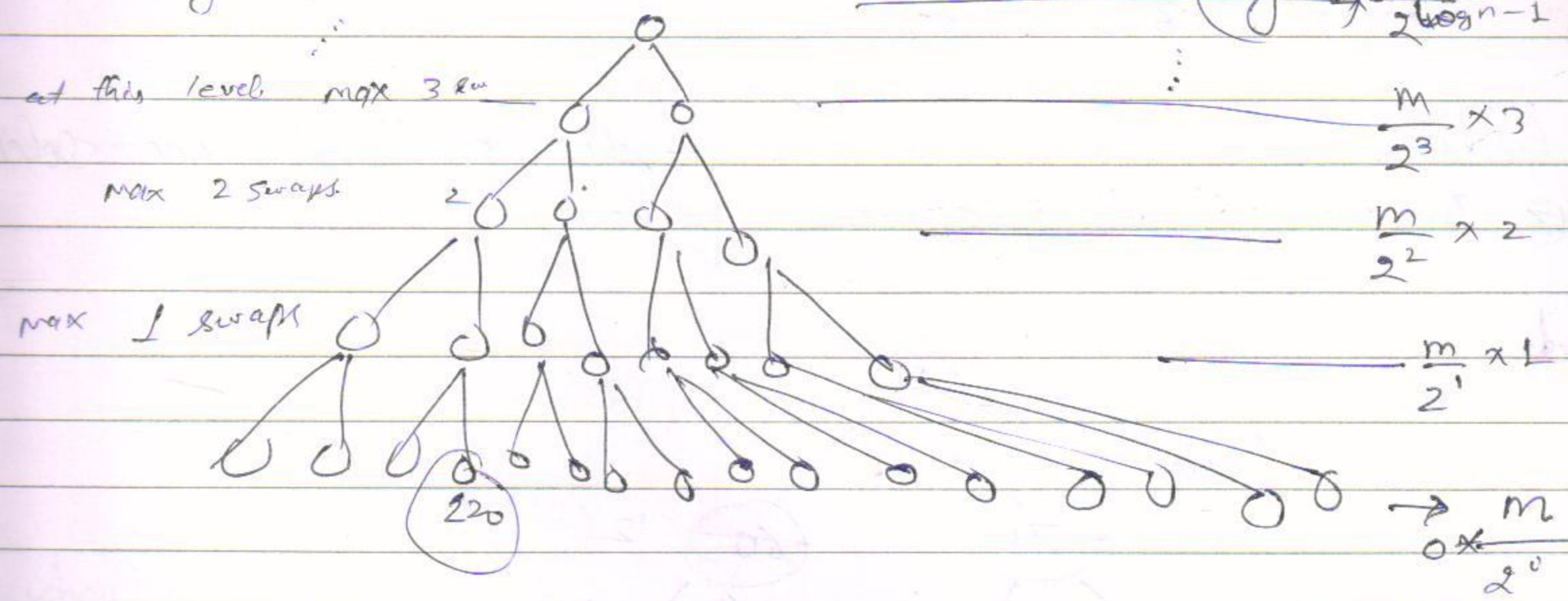
$\lfloor 3.5 \rfloor = 3$ internal nodes

$\lceil 3.5 \rceil$ = 4 leaf nodes

$n = 31$

$m = \frac{n}{2} = 16$

Date



Time complexity of Build heap is -

$$T(B\text{-}Heap) = \frac{m}{2^0} \times 0 + \frac{m}{2^1} \times 1 + \frac{m}{2^2} \times 2 + \cdots + \frac{m}{2^{\log n - 1}} \times \log n -$$

$$= m\left[\frac{0}{2^0} + \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{\log n - 1}{2^{\log n - 1}}\right]$$

↳ AP & GP series. Combination of AP & GP.

$$= m\left[\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^{\log n - 1}}\right]$$

$= m * O(1)$

$= m$

$= \frac{n}{2} \Rightarrow \underline{\Theta(n)}$ $\because$ $m = n/2$

Hence Build heap method takes $\Theta(n)$ times.

Date

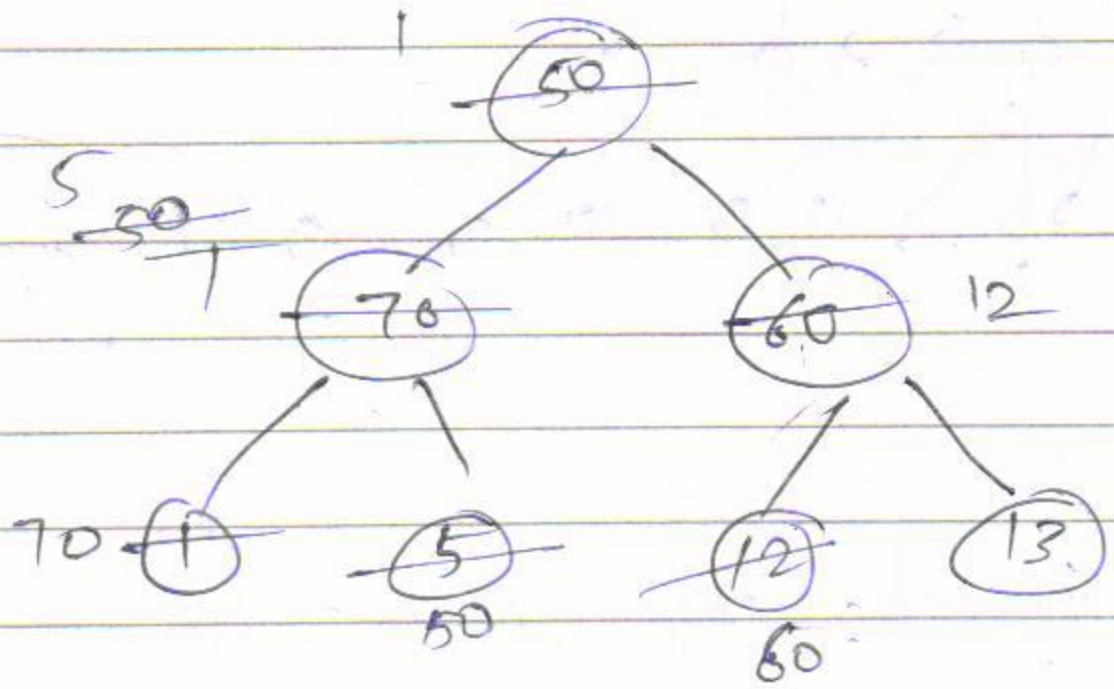
Q.(i) Construct Min heap for the following elements.

50, 70, 60, 1, 5, 12, 13

(2) From the above min heap — show result after one Delete

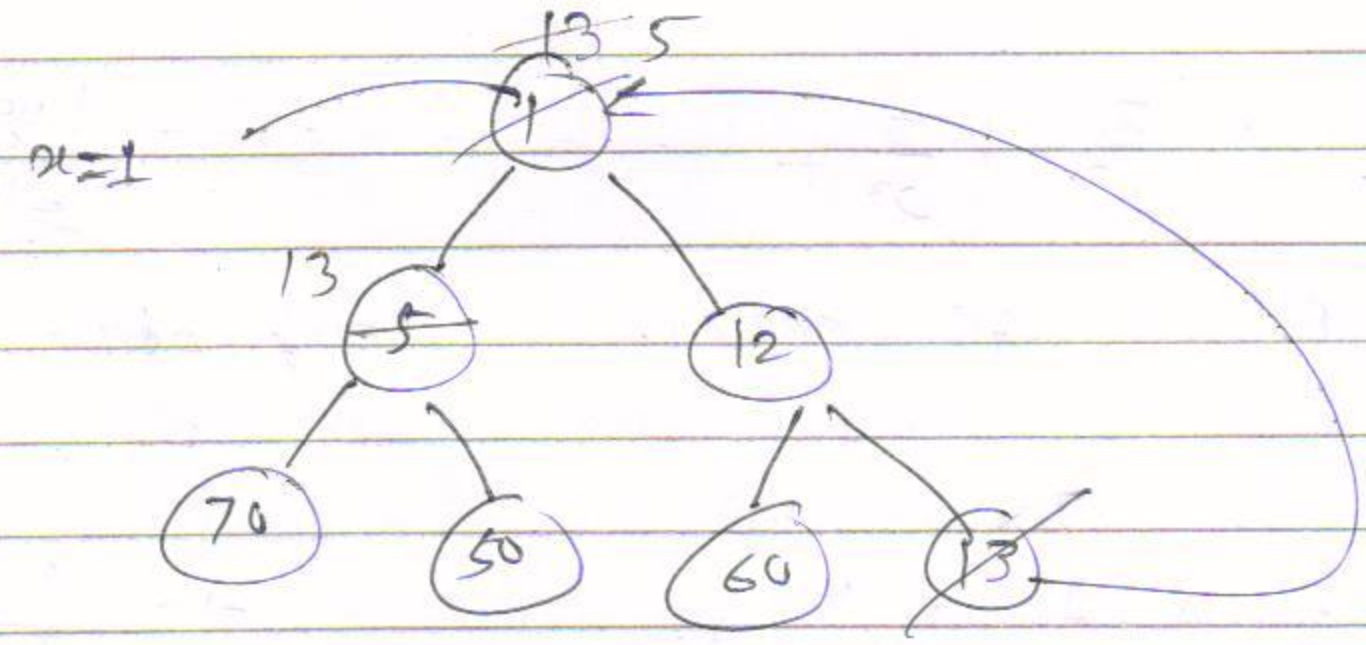
(3) Insert 5 into above min heap.

Ans.

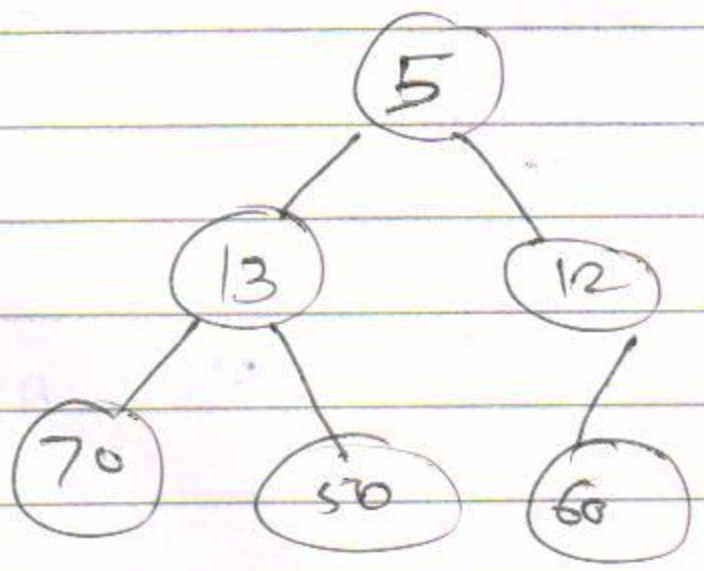


| 1 | 5 | 12 | 70 | 50 | 60 | 13 |

Ans (i)



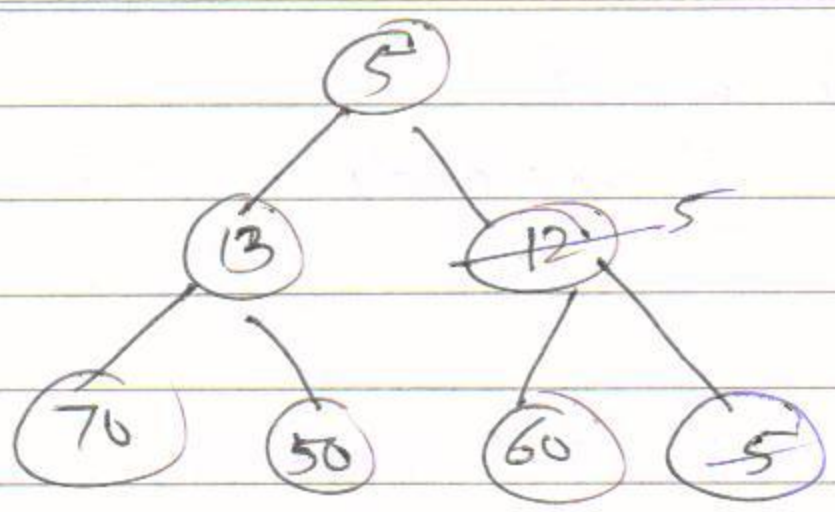
(ii)



Ans. (ii)

(iii) insert 5 -

Date

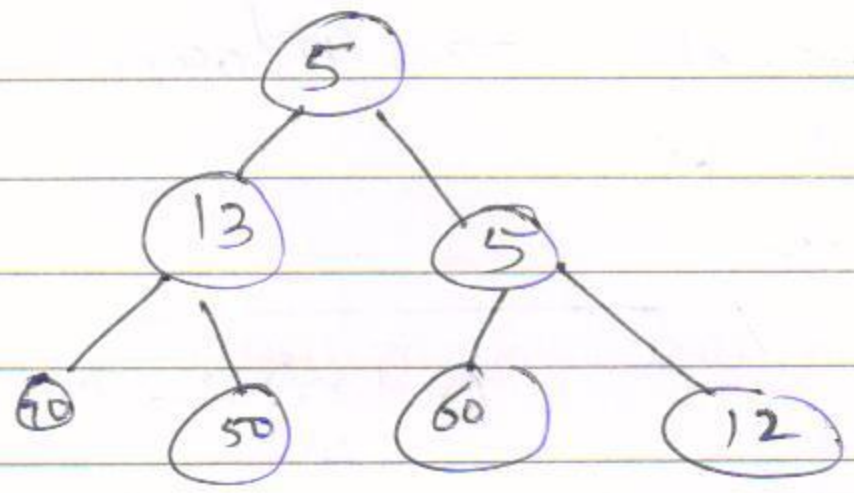


→ equal no need to swap.

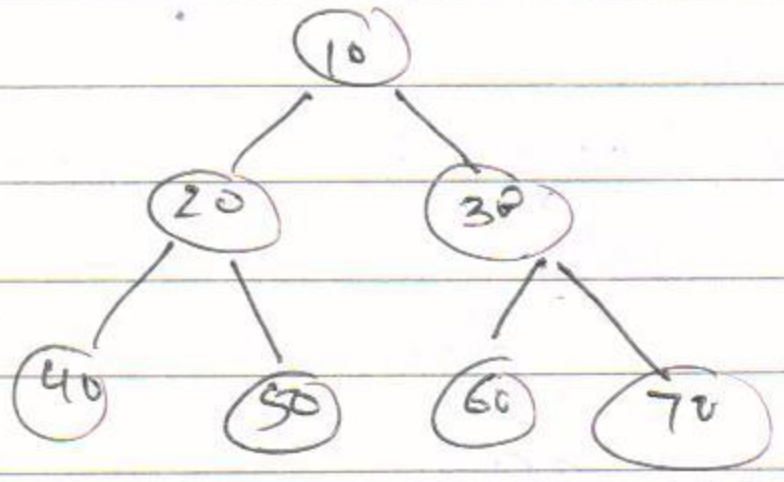
5

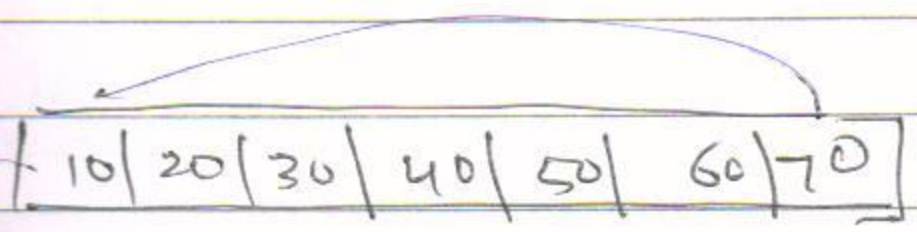
12

Ans (iii)

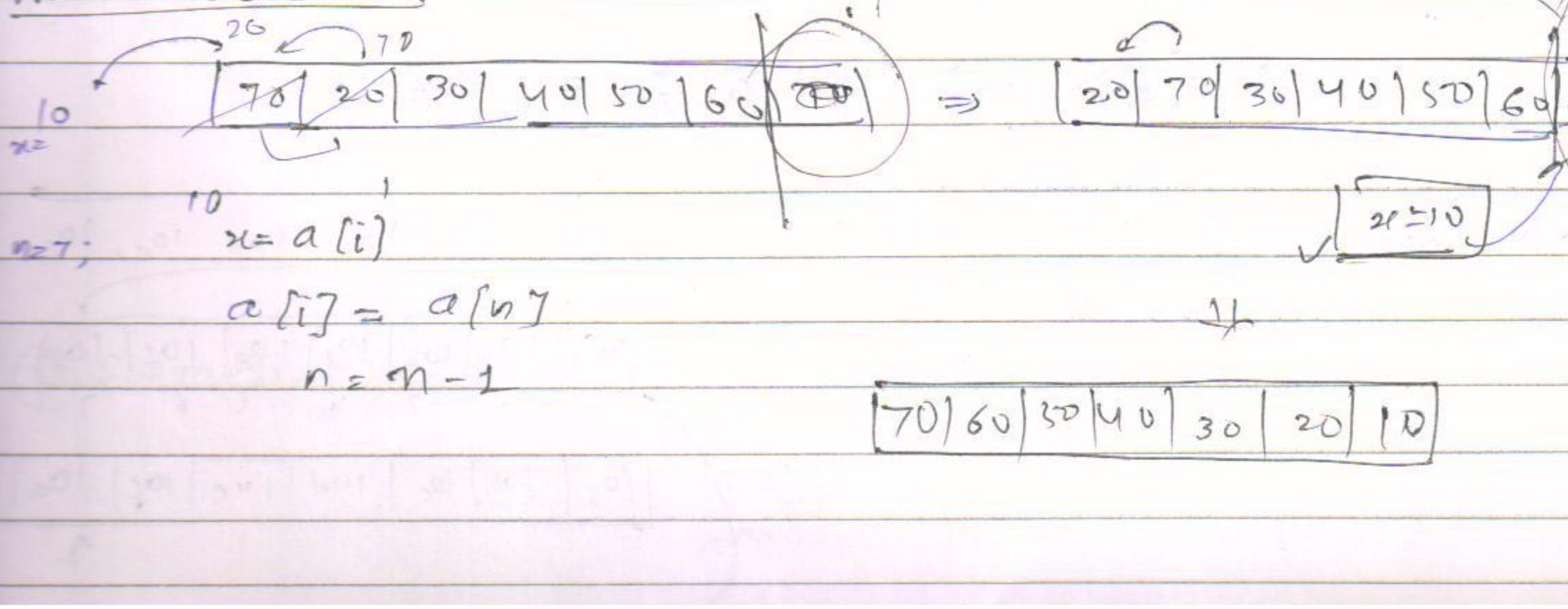


Heap Sort :-





Now delete 10 :-



Date

⇒ Using Min heap if we try to sort the array then it will give descending orders.

Algo :-

① Creating Minheap ⟹ O(n)

② Delete one by one and store from right to Left ⇒ O(nlogn)
Deleting one element (logn)
and for n elements

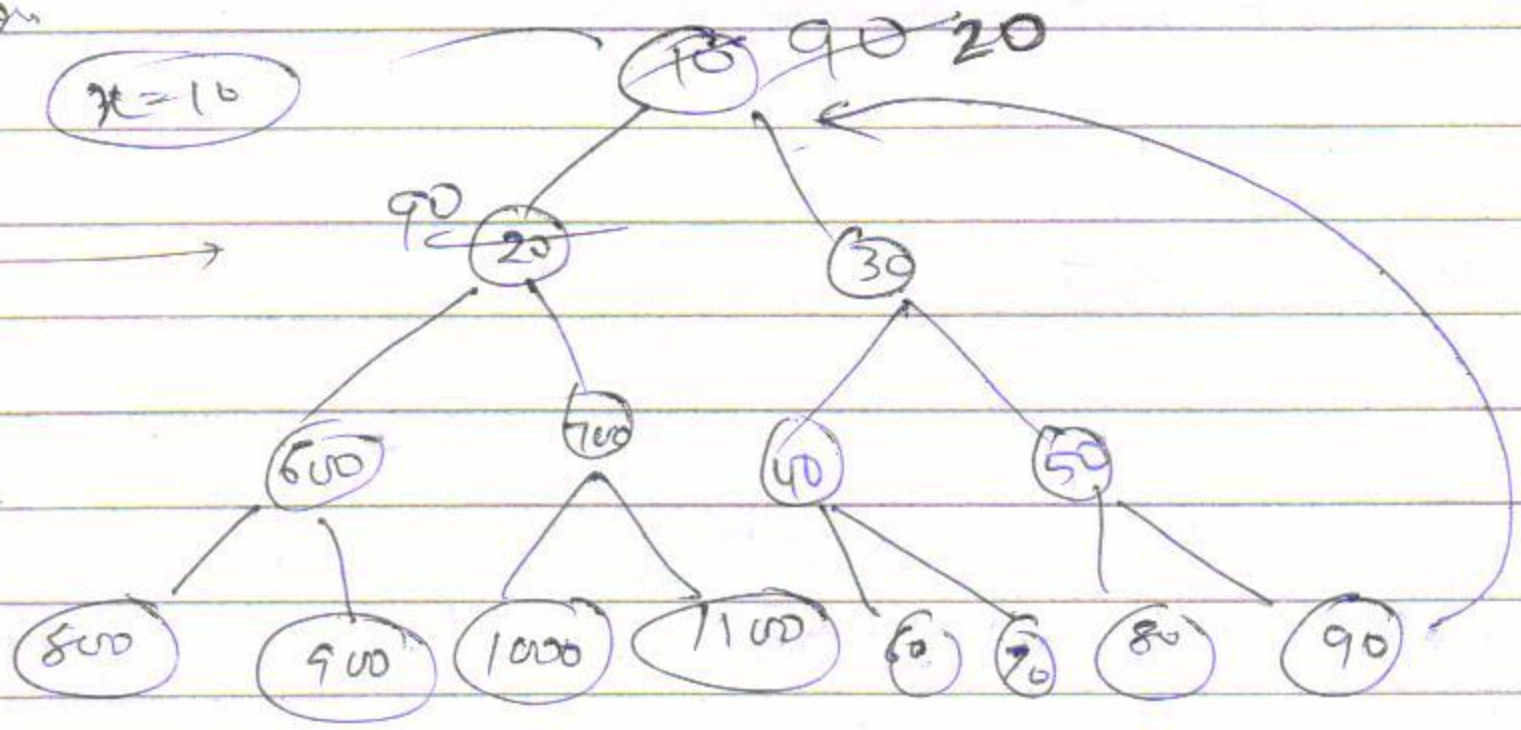
O(n) + O(nlogn) ~

TC → ⟹ O(nlogn)

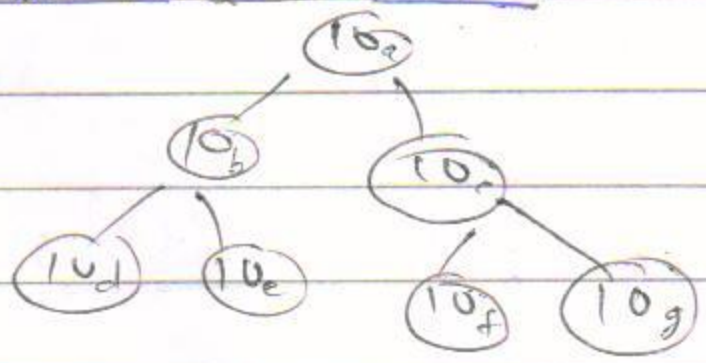
so — in any case same

NOTE :- Sorting through min heap is inplace.

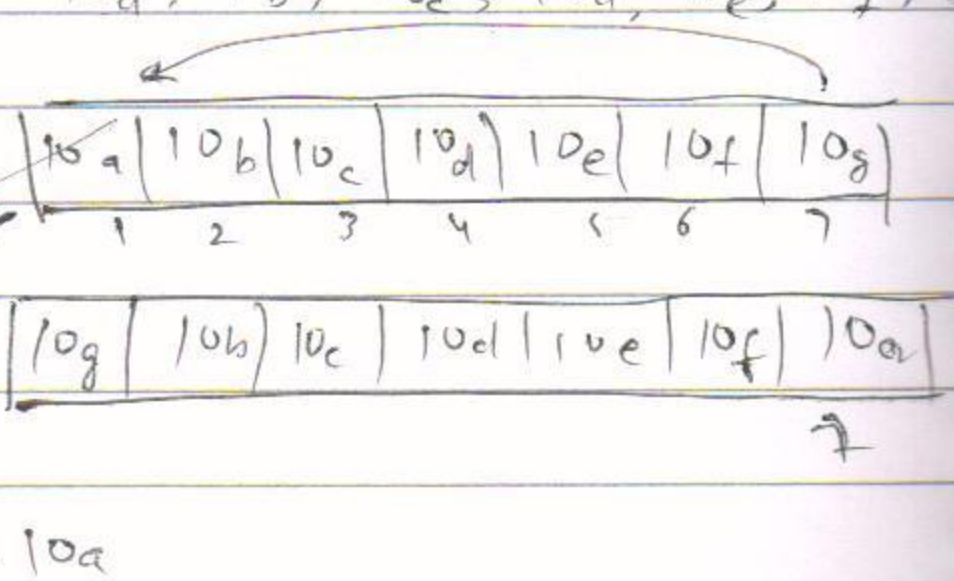
⇒ In best case deleting an element from Min heap is O(1) explaination :-



⇒ Min heap sort is Not stable :-



⇒ 10a, 10b, 10c, 10d, 10e, 10f, 10g



Date

⇒ | 10_b | 10_c | 10_e | 10_d | 10_e | 10_f | 10_g | 10_q |

Note :- ~~Not~~ not stable (Min heap sort is not stable).

⇒ To sort n-element Heapsort will take $\theta(n)$
{ if all elements are same }.
if all elo are same then for B.C = W.C = A.C = $\theta(n)$

Q. finding 1st minimum in min heap = $O(1)$
7 th Min = $O(1)$
1000 th Min = $O(1)$

~~finding~~ Extracting / Deleting 1st element from Min heap = $\log n$
Delete 7 th elel
1000 th

$7 \times \log n = \log n$
$\Rightarrow 1000 \log n$
$= (\log n)$

⇒ Min heap

① Creating Min heap ⇒ $O(n)$
② finding 1st Min ⇒ $O(1)$
2nd Min ⇒ $O(1)$
3rd Min ⇒ $O(1)$
⋮
1000 : Min ⇒ $O(1)$
nth Min ⇒ $O(n)$
③ Extracting / Deleting 1st min ⇒ $O(\log n)$
2nd min ⇒ $O(\log n)$
⋮
1000th Min ⇒ $O(\log n)$
⋮

Date

Sorting Algo

① Comparision based sorting Algos	BC	WC	AC
1. Bubble sort	n^2	n^2	n^2
2. Merge sort	$n\log n$	$n\log n$	$n\log n$
3. Quick sort	$n\log n$	n^2	$n\log n$
4. Heap sort	$n\log n$	$n\log n$	$n\log n$
5. Selection sort	n^2	n^2	n^2
6. Insertion sort	n	n^2	n^2

② Non - comparision Based sorting algo

① Counting sort
② Radix sort
③ Bucket sort
} $\Theta(n)$

Q. → In all ~~worst case~~, or in all worst one is $O(n\log n)$

A. → In all best case is $\alpha\ n^2$

compassion based sorting techniques. case T. complexities best

T. Complexities, wrost one.

Comparision based sorting techniques —

Sorting algo	Best C.	Worst case	Average C.
1. Bubble Sort	→ n^2	n^2	n^2
2. Merge sort	$n\log n$	$n\log n$	$n\log n$
3. Quick sort	$n\log n$	n^2	$n\log n$
4. Heap sort	$n\log n$	$n\log n$	$n\log n$
5. Selection sort	→ n^2	n^2	n^2
6. Insertion sort	n	n^2	n^2

Date

Radix Sort :- Ex:- 265 734 427 128

passes depend on no of digits in largest number.

① pass 1: 734 265 427 128 ⇒ O(n)

② pass 2: 427 128 734 265 ⇒ O(n)

③ pass 3:- 128 265 427 734 ⇒ O(n)

3*O(n)

O(n)

§ Greedy Technique -

Date

Note:- Most of the problems in greedy contains n - i/p's and our objective is finding a subset which will optimizes our goal.

→ output should cotain strict subset.. means all element should not be there in output or only one element should be there..

→ Maximizes the Profit and minimizes cost

Basics:-

1. Solution Space

eg→ all posibility, means if 250 students are there we want 10 only then all combinations of 10, ⇒ $^{250}C_{10}$

⇒ Set of all posible solutions with the given n no. of inputs is called solution space.

so solution space ⇒ $(10_a, 10_b, 10_c, 10_d \dots\dots {}^{250}C_{10})$.

2. Feasible solution :- It is one of the solution from solution space which will satisfy our condition.

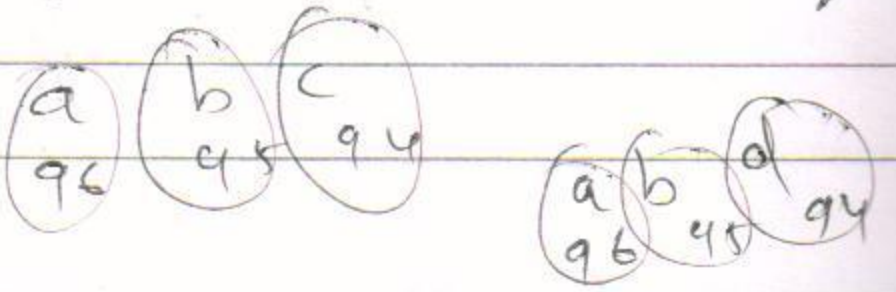
Feasible solution ⇒ $(10_a, 10_b, 10_c, 10_d \dots\dots {}^{50}C_{10})$

eg:- for 80% (minimum 80%), Let 50 student have 80%

3. Optimal solution :- It is one of the Feasible solution from solution space which optimise our goal.

so only one solution ⇒ 10_a [need not be unique]

eg→ if we went 3 toppers. →



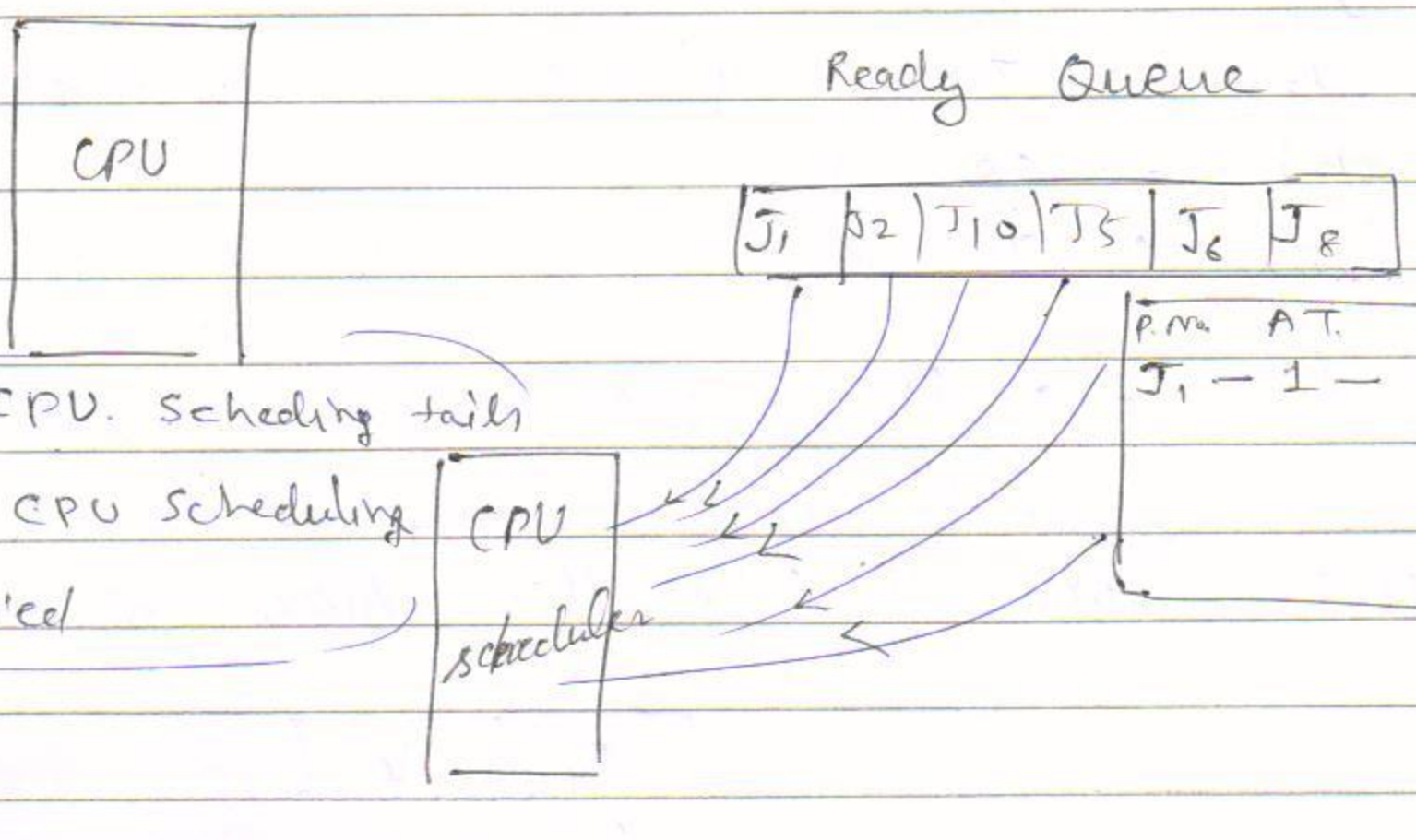
Date

⇒ Applications of greedy :-

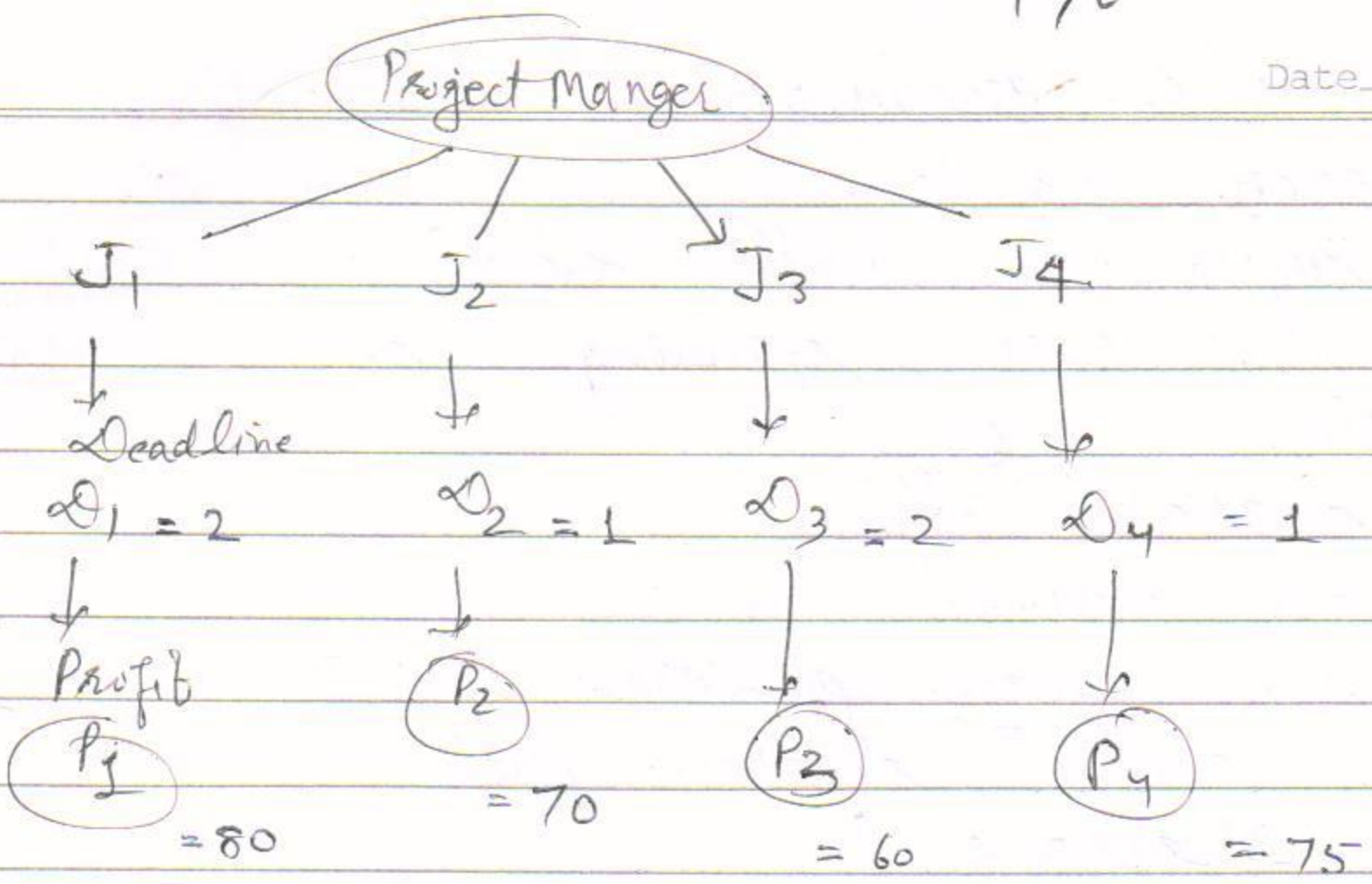
(1.) knapsack problem
(2.) ✓Job sequencing with Deadline
(3) Minimum cost spanning Tree.
  (i) prim's Algo
  (ii) Kruskal Algo
(4) Huffman coding
(5) Optimal merge pattern
(6) single source shortest path.
  (i) Dijkstra's
  (ii) Bellman Ford.

(2) Job sequencing with Deadline :-

(1.) Single Cpu → multiple jobs can't run.
(2) No - interleaving → Round Robin - fails → No-interleaving means Non-pre-empti
(3) Arrival time same → FIFO fails
(4) Every Job requires 1 - unit of time. → SJF fails.



Date



1. { J2 , J1 } ⟹ 150
2. { J2 , J3 } → 130
3. { J2 , J4 } → X Not feasible
4. { J3 , J2 } → X
5. { J4 , J1 } → 155 → only this is optimum
6. { J4 , J3 } → 135
7. { J1 , J3 } → 140
8. { J3 , J1 } → 140

6 options

9. { J1 } → 80
10. { J2 } → 70
11. { J3 } → 60
12. { J4 } → 75

4 option

total 11 feasible sol^n

13. { } → 0 { 1 option

Direct Solution :- (1) take the Max. Deadline -

J4	J1
1	2

deadline — deadline.

Profit - 80 + 75 = 155

(2) → start from Right hand.

Q. $n = 5$ Date

Jobs =	J_1	J_2	J_3	J_4	J_5
profits =	80	25	90	75	15
deadline =	2	1	3	1	2

Soln :-

J_4	J_1	J_3
75	80	90
1	2	3

$\Rightarrow$ 245 optimal.

b. J_4 , J_1 , J_3 $\longrightarrow$ Max Profit

75 80 90 245

Q. $n = 7$

Jobs =	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Profit $\Rightarrow$	25	75	15	85	12	7	80
deadline $\Rightarrow$	2	3	4	1	6	4	3

b.

J_4	J_2	J_7	J_3		J_5
1	2	3	4	5	6

85 75 80 15 12 $\Rightarrow$ 267

J_4 J_2 J_7 J_3 J_5 =

① Sort the jobs in decreasing order of profits.

J_4	J_7	J_2	J_1	J_3	J_5	J_6
85	80	75	25	15	12	7
1	3	3	2	4	6	4

no need to go further

Gate'

Date

Q. n = 9

Job :	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
Profit :	25	35	70	12	21	17	5	55	42
dead :	5	2	7	1	4	2	3	1	2

A.

① sort the array in decreasing order of ~~array~~ profit.

J_3	J_8	J_9	J_2	J_1	J_5	J_6	J_4	J_7
70	55	42	35	25	21	17	12	5
7	1	2	2	5	4	2	1	3

J_8	J_9	J_7	J_5	J_1		J_3
1	2	3	4	5	6	7

55 42 J_7 21 25 70 = 218

A

A. J_8 J_9 J_7 J_5 J_1 J_3

The jobs left out are counted as panelty.

35 + 17 + 12 = 64 panelty.

① int a[5]

② struct node

{

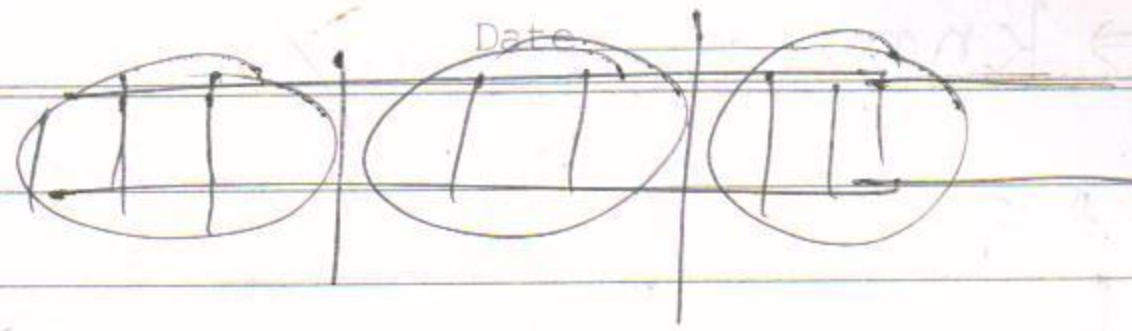
char Job [3]; → 3 B

int profit; → 2 B

int deadline; → 2 B

} struct node a;

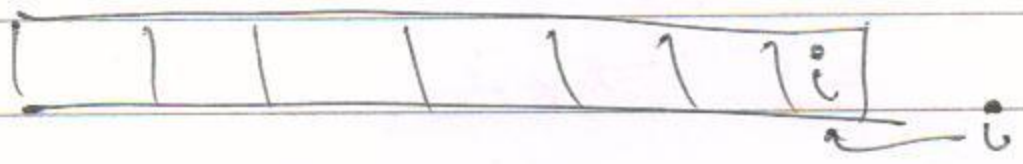
struct node a[5];



① sort the array based on profit. in decreasing order. → O(nlogn)

② Finding Max deadline → Maximum deadline.

Take an array of and start from the Right hand side.



O(n finding max deadline

③ For everyslot i, apply linear search to find a job which contain deadline ≥ i

n × n = O(n²)

so O(nlogn) + O(n) + O(n²)

= O(n²). ✓

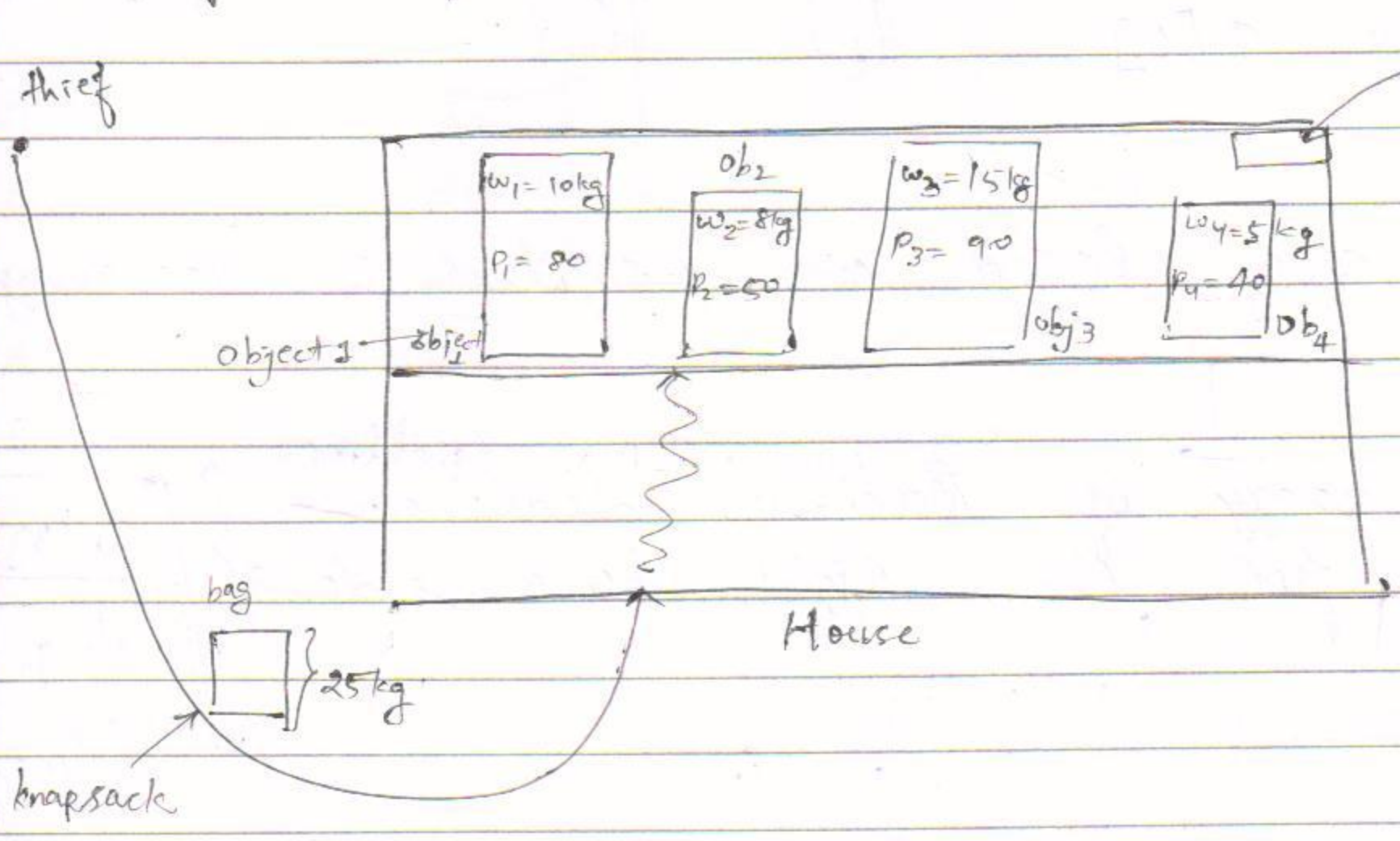
⇒

6-Nov-12

200

Date

⇒ knapsack - problem :-



I can take = $(1*10) + (1*8) + \frac{7}{15} \times 15 + 0 \times 5 = 25$

$= 1*80 + 1*50 + \left(\frac{7}{15} \times 90\right) + 0*40 =$

$= 80 + 50 + 42 + 0 = 172$ (Feasible soln)

if ans is not cause of damage of knapsack then ans/soln is Feasible.

→ M = knapsack size.

obj	ob_1	ob_2	ob_3	----	ob_n
weight	w_1	w_2	w_3	----	w_n
profit	p_1	p_2	p_3	----	p_n

① Problem :- $\sum_{i=1}^{n} w_i > M$

② feasible :- $\sum_{i=1}^{n} x_i w_i \leq M$ — M is knapsack size

where x_i = fraction of weight { $0 \leq x_i \leq 1$ }

Date

(3) optimal : $\sum_{i=1}^{n} x_i \times P_i$ is maximum.

Q. $n = 3$ M = Capacity of knapsack = 20

obj :	ob_1	ob_2	ob_3			
profits :	25	24	15	$\frac{25}{18}$	$\frac{24}{15}$	$\frac{15}{10}$
weight :	18	15	10			

Ans.

$1 \times 24 + \frac{5}{10} \times 15 = 24 + \frac{15}{2} = 7.5 = 31.5$ profit

$1 \times 15 \quad \frac{5}{10} = 25$ weigh

Greedy about profit and weight both.

obj 1 : $18 \Rightarrow 25$

$1 \longrightarrow 25/18 = 1.3$

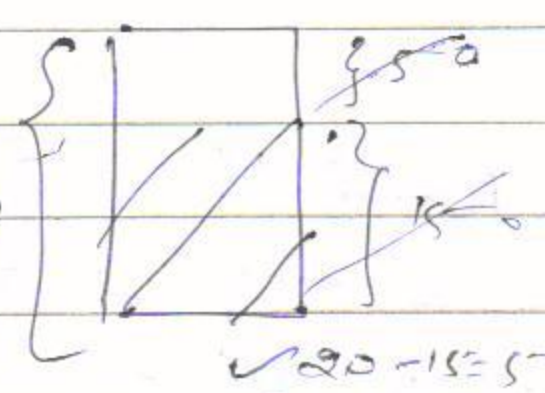
obj. 2 : $15 \longrightarrow 24$

$1 \longrightarrow 25/15 = 1.6$

obj. 3 : $10 \longrightarrow 15$

$1 \longrightarrow 15/10 = 1.5$

$\left(\frac{0}{x_1} \quad \frac{1}{x_2} \quad \frac{5/10}{x_3}\right)$



$\Downarrow$

Feasible $= 0 \times 18 + (1 \times 15) + \frac{5}{10} \times 10$

$= 0 + 15 + 5 = 20$ weight feasible

Profit $= 0 \times 25 + (1 \times 24) + \frac{5}{10} \times 15 =$

$= 0 + 24 + 7.5 = 31.5$ profit

202

Q. $n = 5$ $M = 25$

obj:	ob_1	ob_2	ob_3	ob_4	ob_5
Profit:	15	30	7	25	10
weight:	9	2	8	12	4
Ratio	1.6	15	.9	2.0	2.5
		1		3	2

$P = 30 + 10 + 25 + \frac{7}{9} \times 15 =$

$W = 2 + 4 + 12 + \frac{7}{9} \times 9 = 25 = M$ ✓

$P = 65 + \frac{35}{3} = 11.66$

$P = 76.66$

23
2

$25 - 2 = 23$

$23 - 4 = 19$

$19 - 12 = 7$

Q. $n = 7$ $M = 28$

obj =	ob_1	ob_2	ob_3	ob_4	ob_5	ob_6	ob_7
pro	40	22	35	15	50	25	10
	5	2	7	4	10	8	3
	8	11	5	3.75	5	3.125	3.3
	2	1	3		3		

$W = 2 + 5 + 7 + 10 + 4 = 28 = M$ ✓

$P = 22 + 40 + 35 + 50 + 15$

$= 162$ A

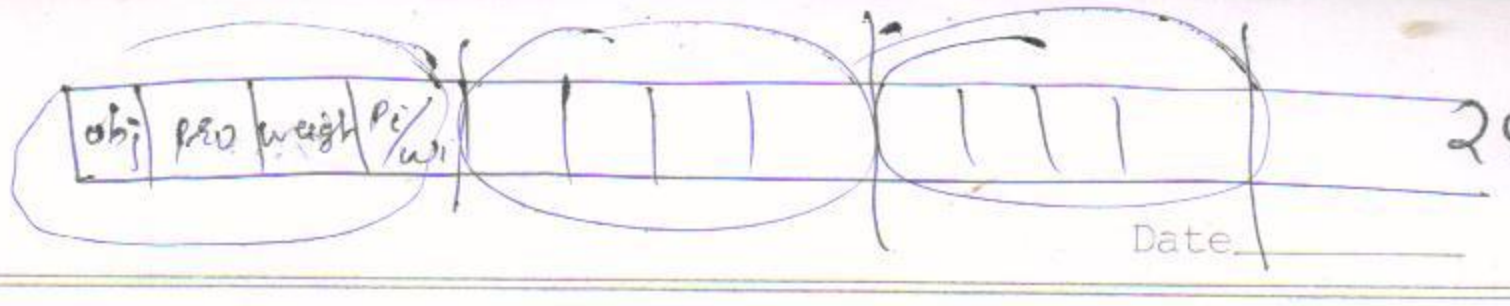
$28 - 5$

$= 23 - 2$

$= 21 - 7$

$= 14 - 10$

$= 4$



Date

Algo:-

① for (i=1 to n) } → O(n)
a[i] = Pi/wi

② Sort the above array based on Pi/wi value in decreasing order. } → O(n log n)

③ Take one by one obyed untill capacity of knapsack becomes zero. } → O(n)

O(n log n)

Q. If time complexity of knapsack problem if all the objects are already given in the decreasing order of their Pi/wi.

Ans No need for step ① & ②
so only 3rd step so O(n). Ans.

NOTE:-

Fractional knapsack:-

Solution space and feasible solutions are ∞. (infinite)

∞ ... 1/3 ... 1/2

(0/1 2 3 4 5 6 7)

Date

8. Huffman Coding:-

① Encoding
② Data Compression.

Message = 100 characters out of 100 — frequencies

a = 55 b = 4
c = 8 d = 1
e = 30 f = 2

ASCII

a — 55 * 8
b = 4 x 8
c = 8 x 8
d = 1 x 8
e = 30 x 8
f = 2 x 8

⇒ :

Sender ——————→ Receiver.

∴ 100 ⇒ 800 bits.

average = 800/100 = 8 bits

// can you

only 6 different character,
so only 3 bit are required

a — 55 x 3 = (165) 000
b — 4 x 3 001
c — 8 x 3 010
d — 1 x 3 011
e — 30 x 3 — (90) 100
f — 5 x 3 101

300 bits ⟹ 300/100 = 3 bits/char.

int a;
char b
a = b;
↳ implicit typecasting
compiler do own
it own

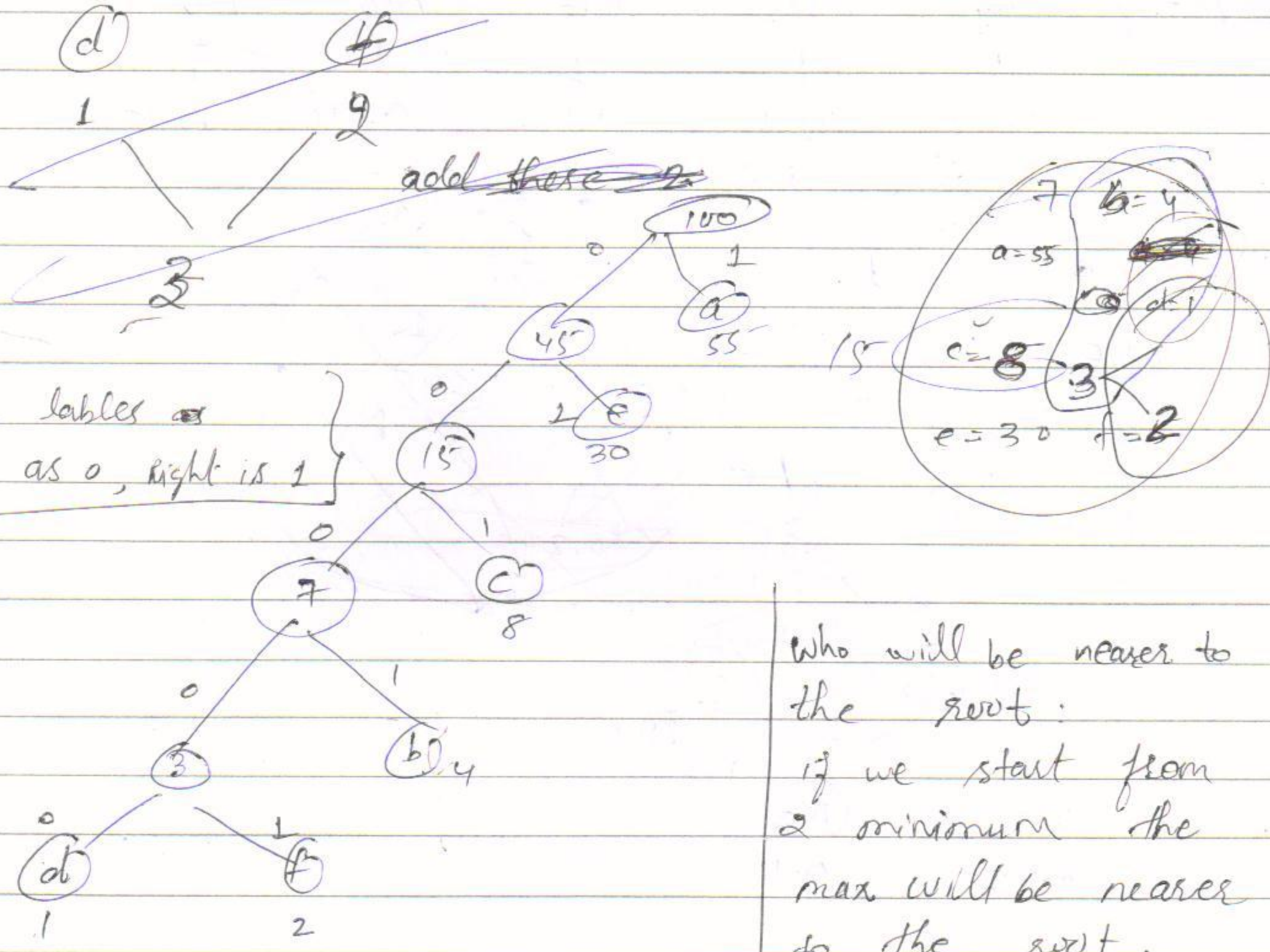
b = a → error.
b = (char) a;
↳ explicit typecasting

//

Huffman
Coding.
//
Non uniform coding.
//
{ More - Frequency = Less bits }

(1) Take 2 less frequency first

Date



Who will be nearer to the root:
if we start from 2 minimum the max will be nearer to the root.
and start giving numbering from root

a = 1

b = 0 0 0 1

c = 0 0 1

d = 0 0 0 0 0

e = 0 1

f = 0 0 0 0 1

Average :- $(55 * 1) + (4 * 4) + (8 * 3) + (1 * 5) + (30 * 2) + 2 * 5$

$= 55 + 16 + 24 + 5 + 60 + 10$

$= 170$ per message

$= \frac{170}{100} = 1.7$ bits per characters.

Note: if Left 0 & Right 1, doing this no answer coming then do again with left 1 & Right = 0

Gate

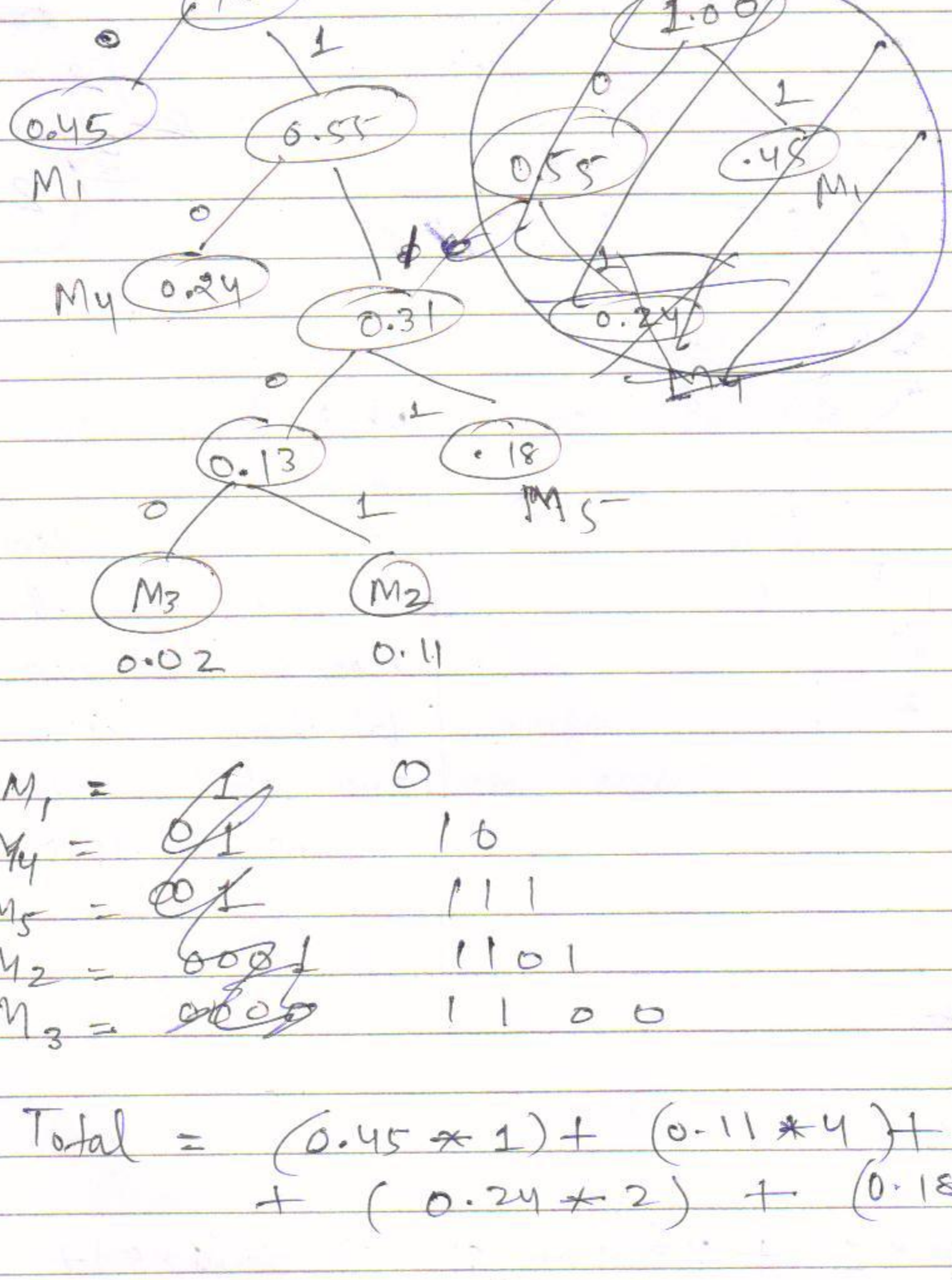
Q.

(206)

Date

$M = (M_1 \quad M_2 \quad M_3 \quad M_4 \quad M_5)$

0.45	0.11	0.02	0.24	0.18



$\Rightarrow M_1 =$ 0

$M_4 =$ 1 0

$M_5 =$ 1 1 1

$M_2 =$ 1 1 0 1

$M_3 =$ 1 1 0 0

② Total = $(0.45 * 1) + (0.11 * 4) + (0.02 * 4) + (0.24 * 2) + (0.18 * 3)$

$= \frac{1.99}{1}$ bits per ~~msg~~ char.

③ ⇒ ENCODED = 1 1 0 0 0 1 1 0 1 1 1 1 0 1 1 0 0 0

decoded ⇒ M_3 M_1 M_2 M_5 M_4 M_3 M_1

assume Left = 1 , Right = 0

Gate

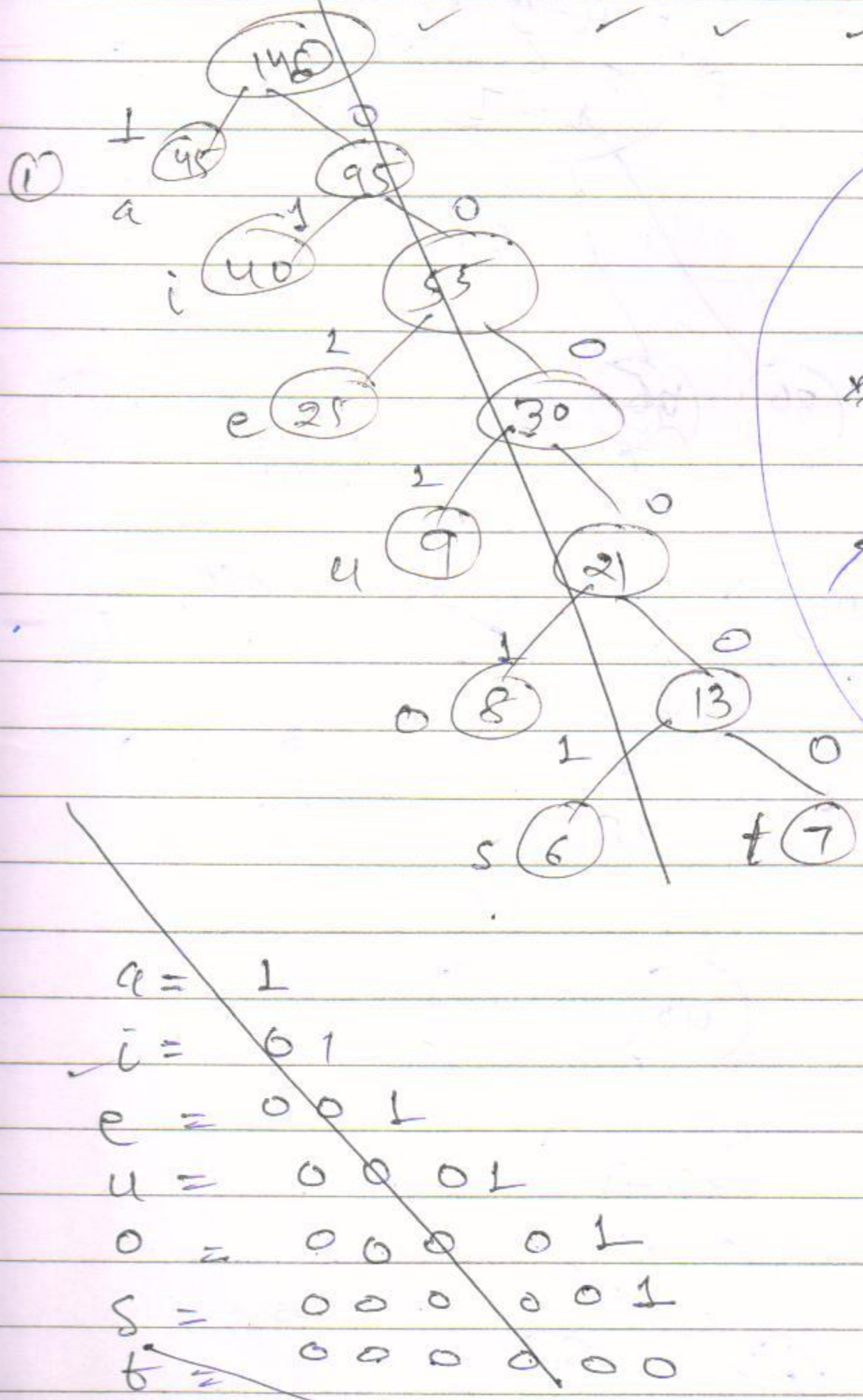
Q.

207

Date

M = (a , e , i , o , u , s , t)

45 | 25 | 40 | 8 | 9 | 6 | 7 ⇒ 140



here you have not taken ~~but~~ always 2 minimum so wrong approach

so see next page

a = 1

i = 0 1

e = 0 0 1

u = 0 0 0 1

o = 0 0 0 0 1

s = 0 0 0 0 0 1

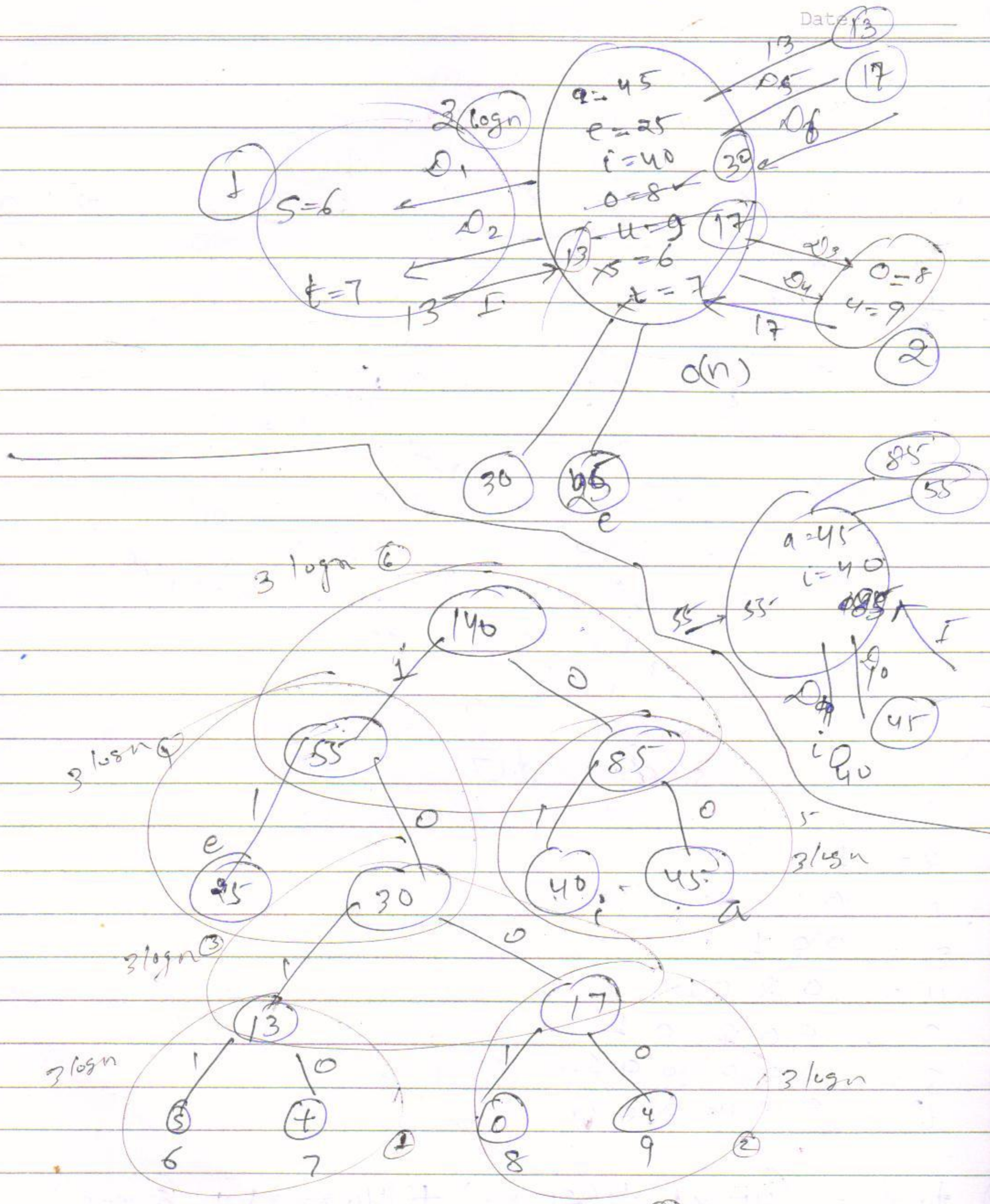
t = 0 0 0 0 0 0

② total = (45 * 1) + (25 * 3) + (40 * 2) + (6 * 5) + (9 * 4) + (6 * 6) + (7 * 6)

= 45 + 75 + 80 + 40 + 36 + 36 + 42

= 4

Date



$$T.C = 6 * 3\log n + n$$
$$= (n-1)\ 3\log n + n$$
$$= n\log n + n$$
$$\underline{TC} = O(n\log n)$$

①
a = 00
e = 11
i = 01
o = 1001
u = 1000
s = 1011
t = 1010

Date

(2) Total = (45 * 2) + (25 * 2) + (40 * 2) + (8 * 4)
+ 9 * 4 + 6 * 4 + (7 * 4)

= 340

$= \frac{340}{148}$ bits / msg.

= 2.43 bits / msg.

⇒ Optimal Merge Pattern :- (2 way Merge Pattern) Date: ____

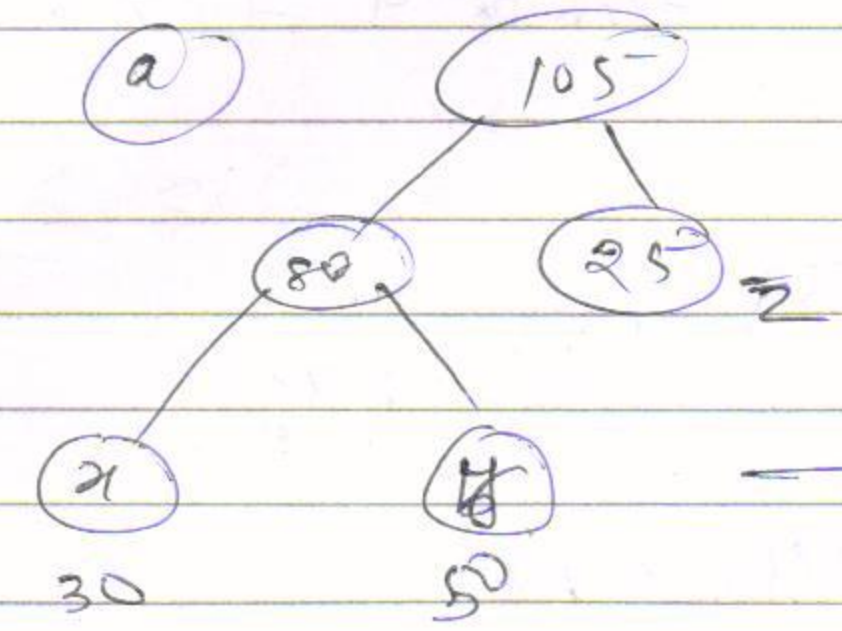
⇒ 3 - files are there. ⇒ file is collection of record.

x = 30

y = 50

z = 25

(a)



105 } = 185 movements
+
80 }

comparision

best case ⇒ 30

worst case = (30+50-1)

= (80-1)

= (n-1)

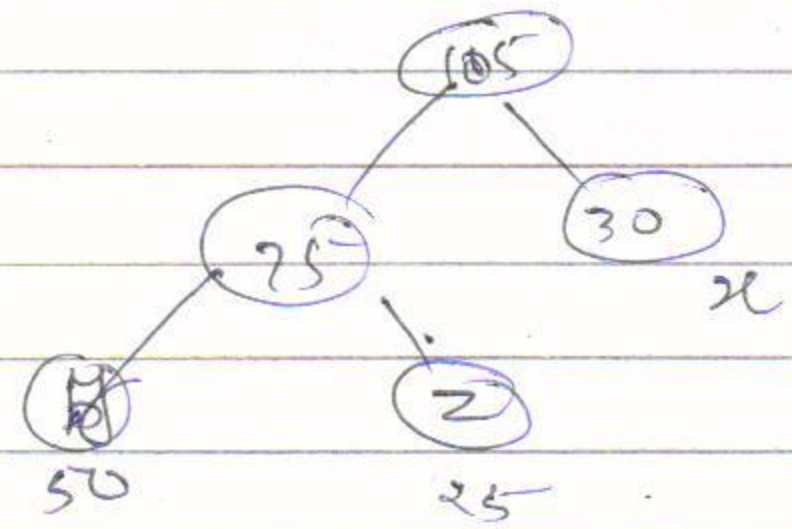
record movements = 80

all are moved

movements always greater so

deciding factor is movements

(b)



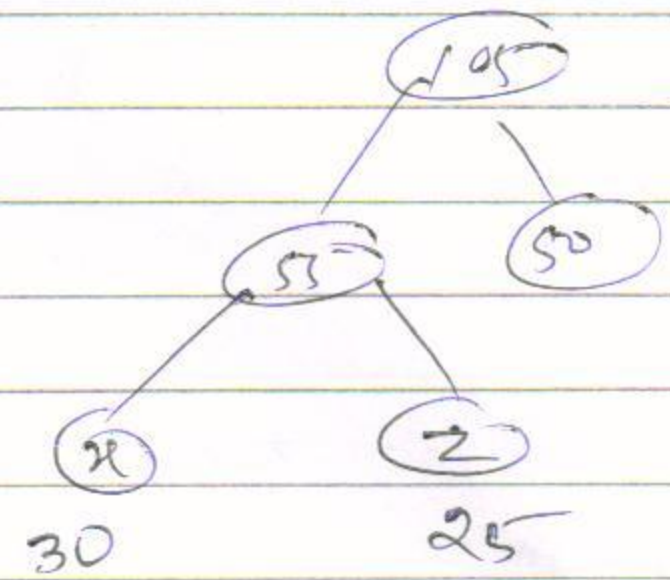
→ 105

+

→ 75

= 105 + 75 = 180

(c)



→ 105

→ 55

105

⇒ + 55

160

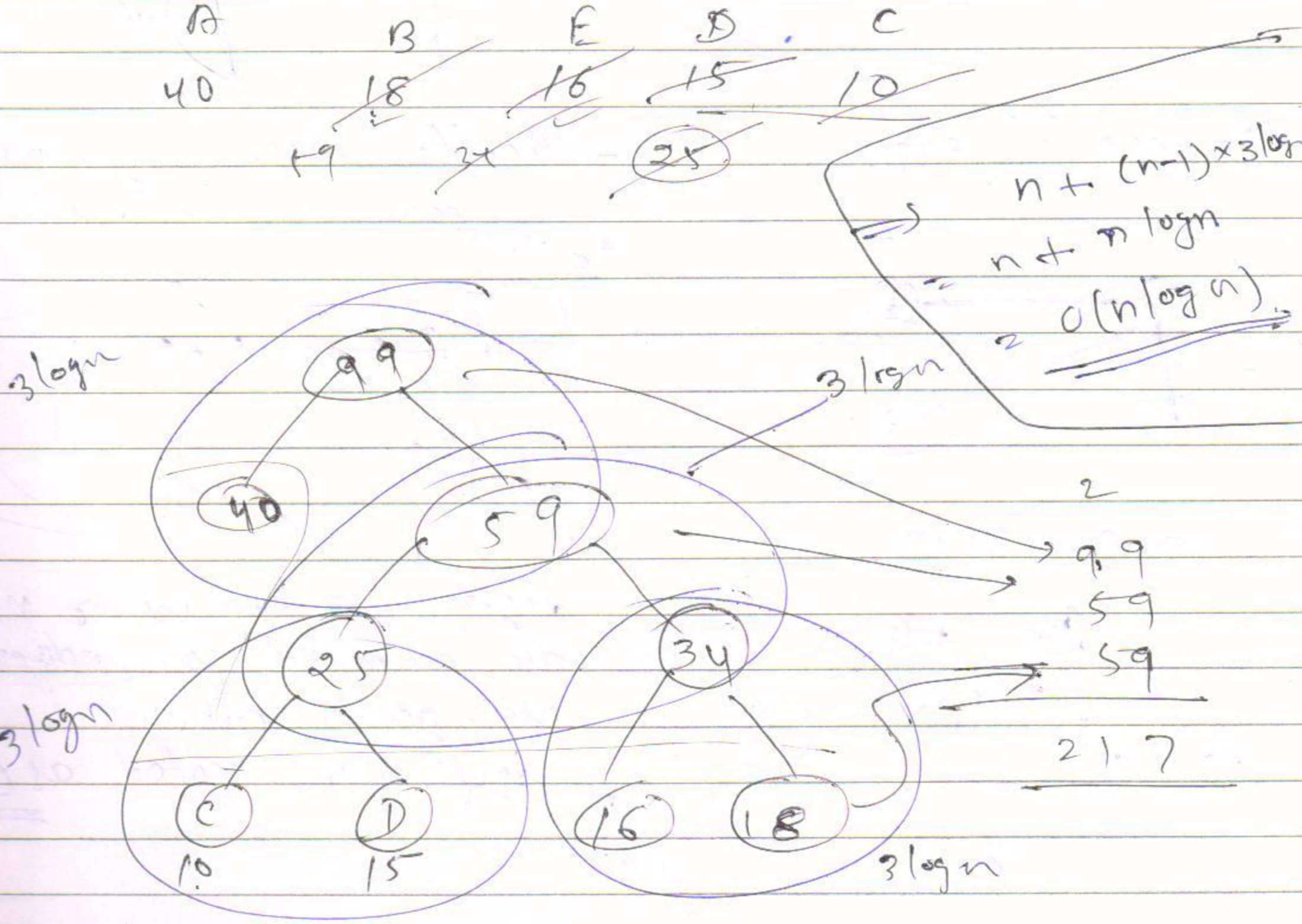
Date:

In above problem there are 3 way of merge pattern in which one is optimal merge pattern and that is 3rd one. (c) beoz that is taking less time.

Q. The minimum no. of record movements are required to merge 5 file..

5 - files ⇒

A	B	C	D	E
↓	↓	↓	↓	↓
40	18	10	15	16

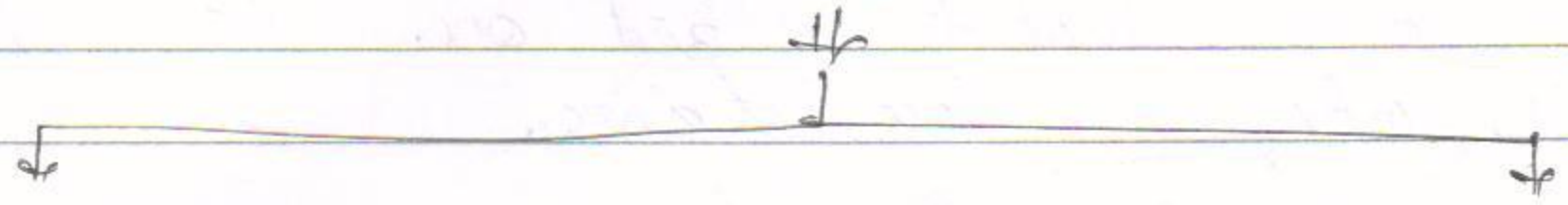


Or Add all internal nodes = minimum no. of record movements = 99 + [illegible] + 59 + 25 + 34

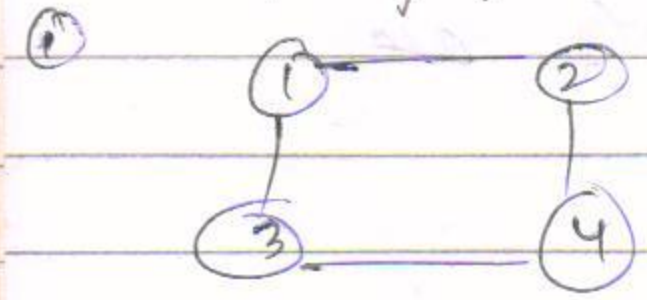
5. Minimum Cost Spanning Tree's-

Date

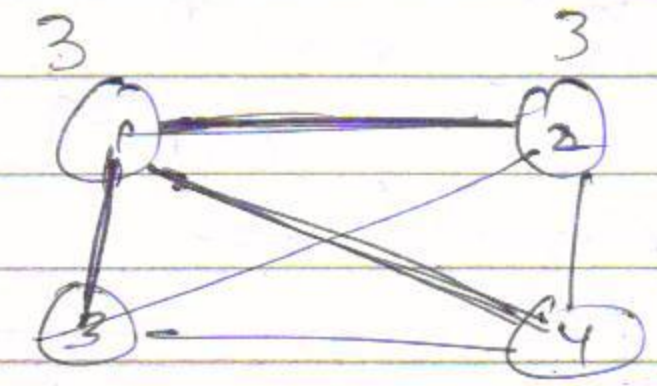
G (V, E)



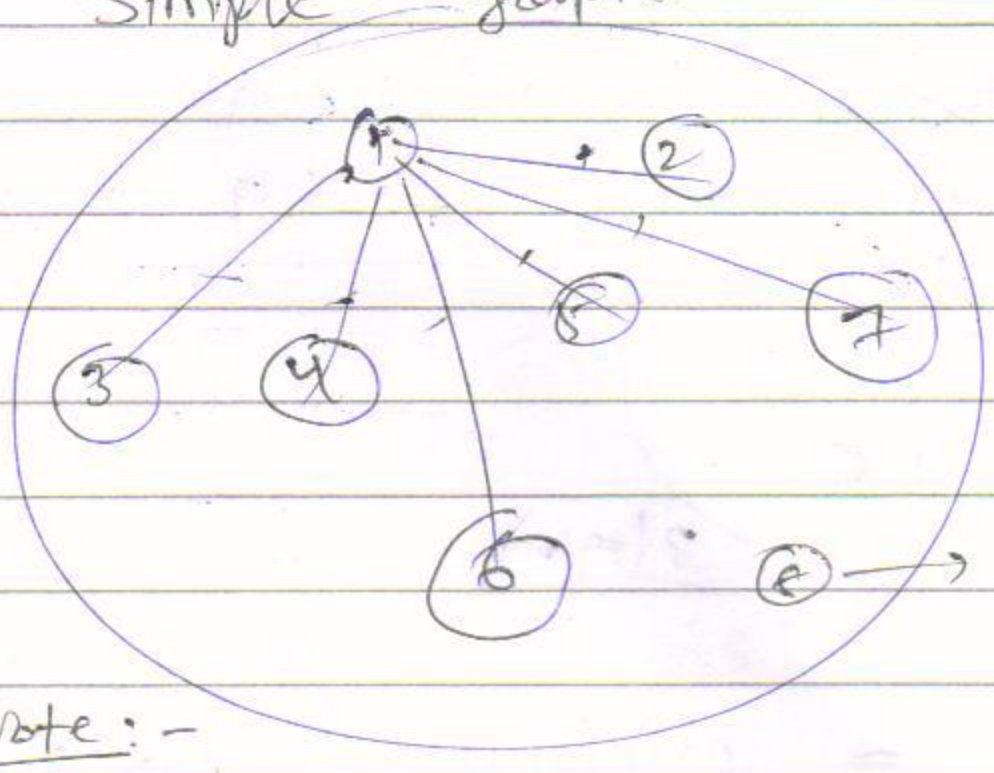
Simple Graph



- No self loop
- No parellel edge



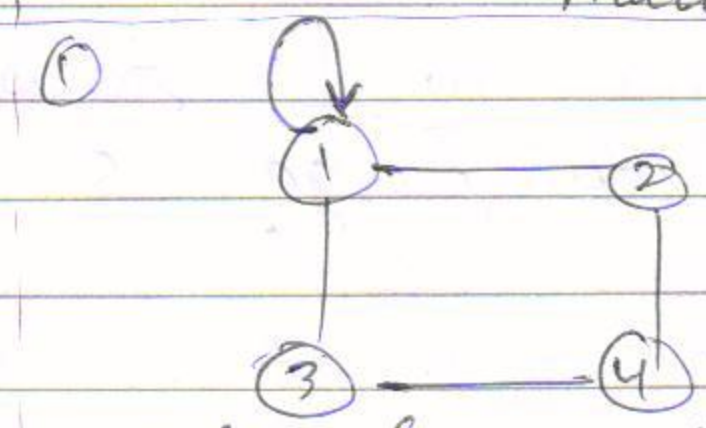
- Simple graph.



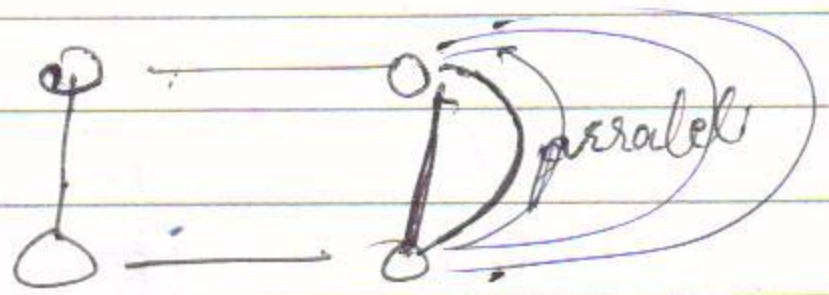
Note :-

In simple graph with n vertices
Max degree of any vertex = (n-1)

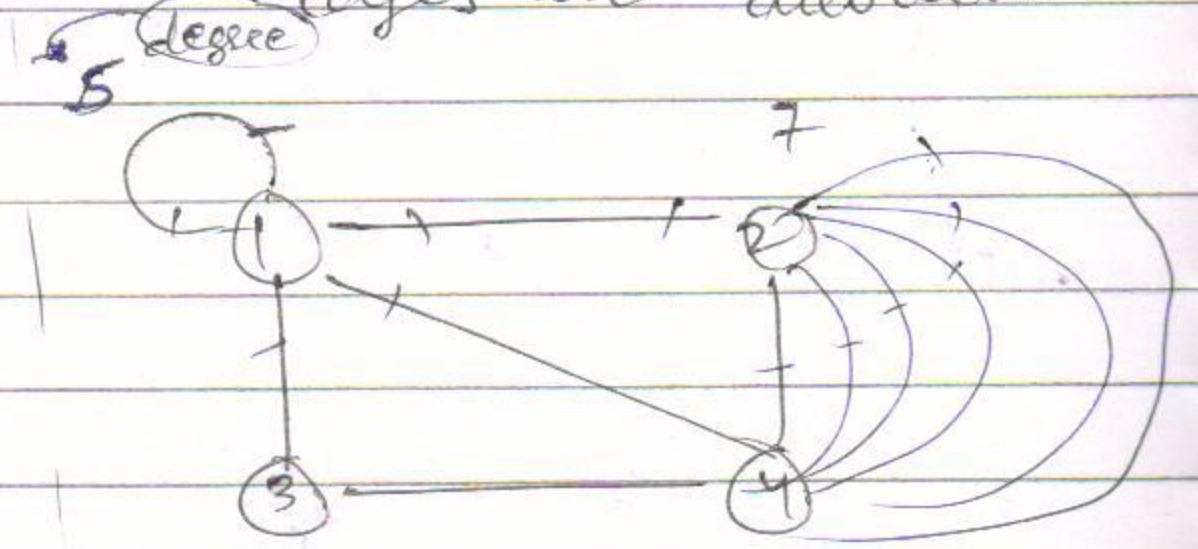
Multi Graph



- self loop allowed



- Parallel edge { any no. of edges are allowed.

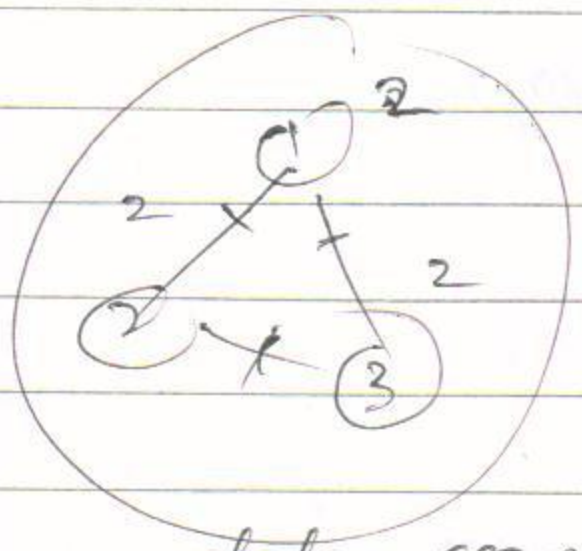


degree is how many nodes are connected or edges are going out.
self loop counted as 2.

Multi-graph Not in syllabus. Date 30 No talk

- Complete graph → every node have max (n-1) degree.

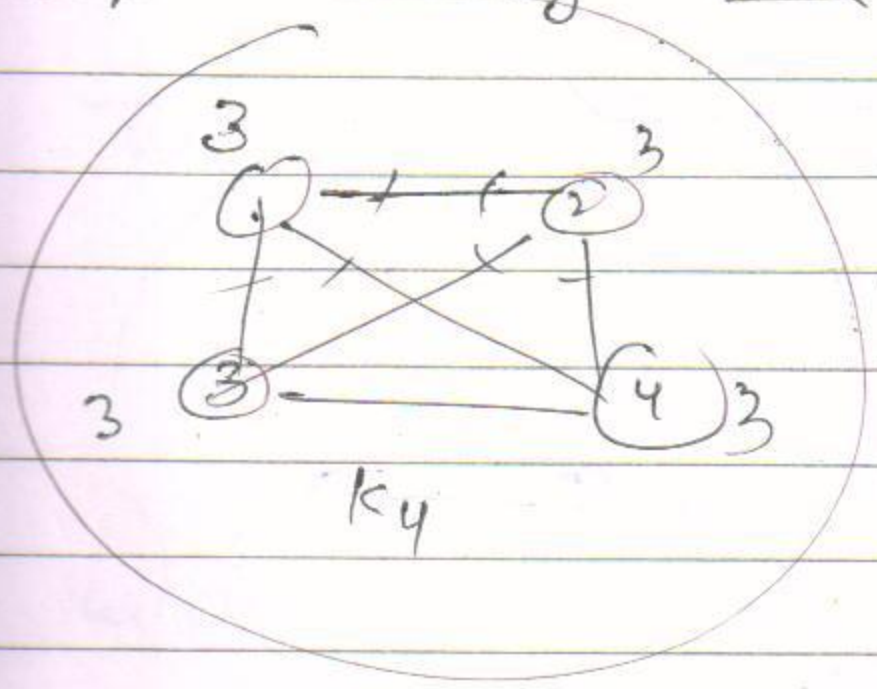
Null graph :- every vertex have 0 degree



K_3

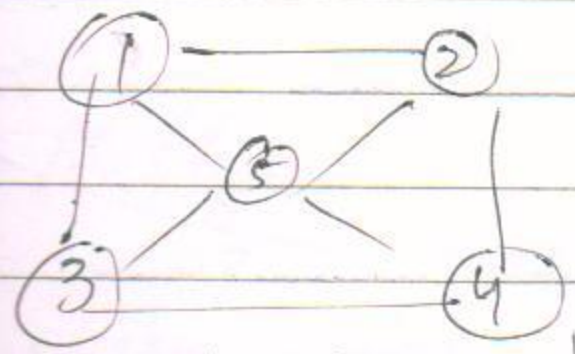
$\Rightarrow \frac{3*2}{2} = \frac{6}{2} = 3$ edges

complete graph.
represented by K_3



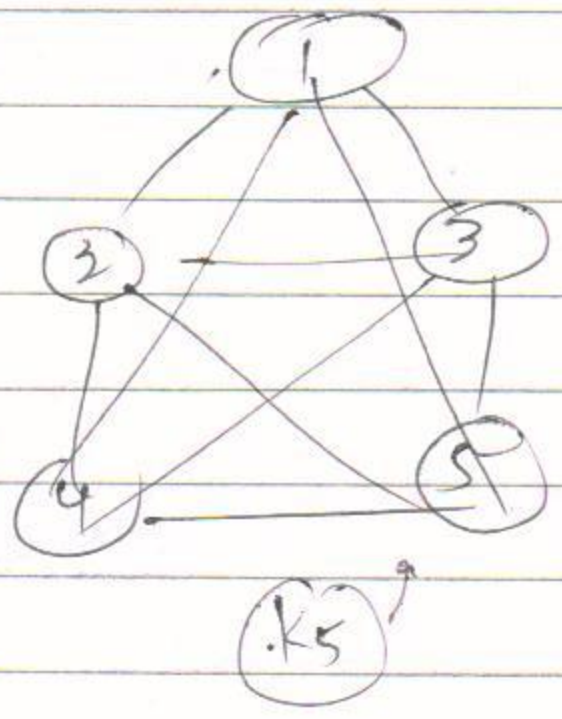
K_4

$\Rightarrow \frac{4*3}{2} = 6$ edges



wheel 5
W_5

wheel graph with 5 nodes



K_5

$\frac{5*4}{2} = 10$ Edges

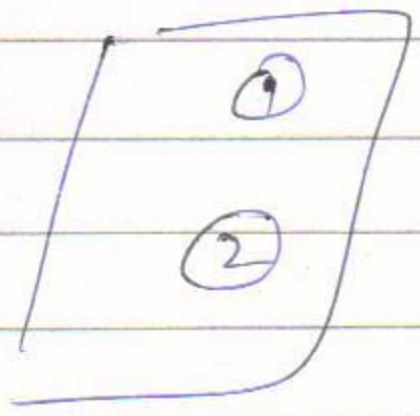
The No of edges possible in n vertex complete graph $= E(K_n) = \frac{n(n-1)}{2}$

Date

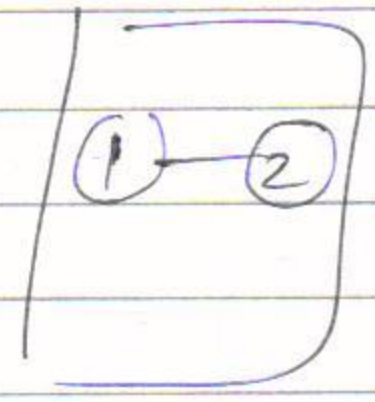
Q

-How many simple graph possible with n vertics :-

n = 2

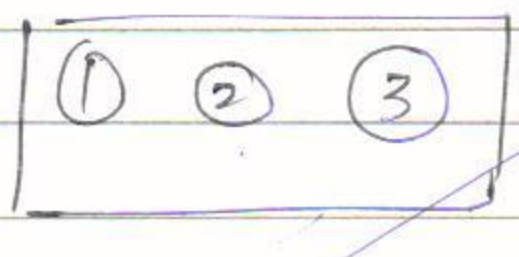


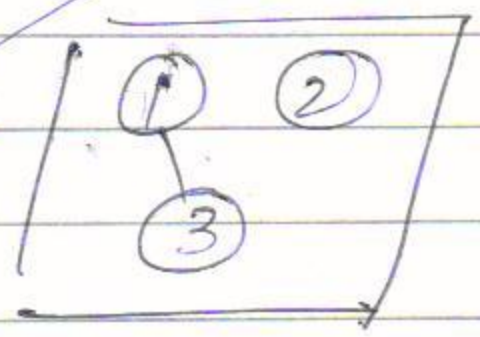
null graph

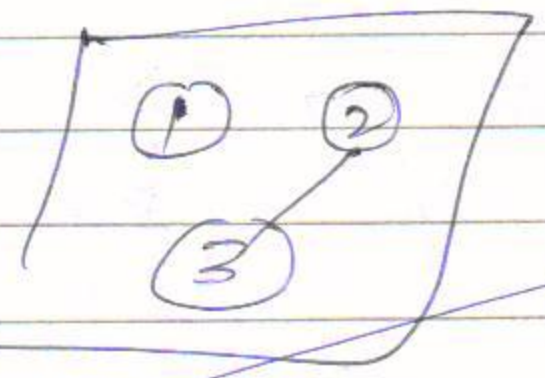


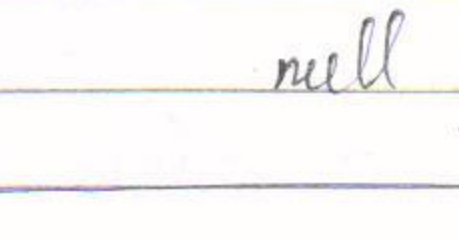
Complete graph

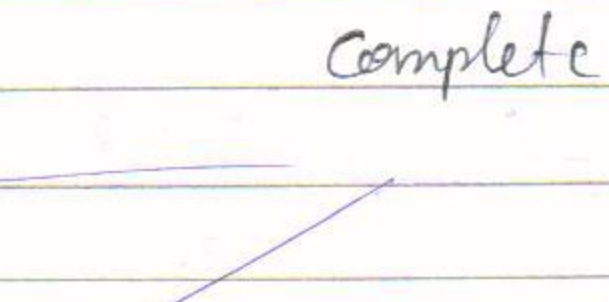
n = 3

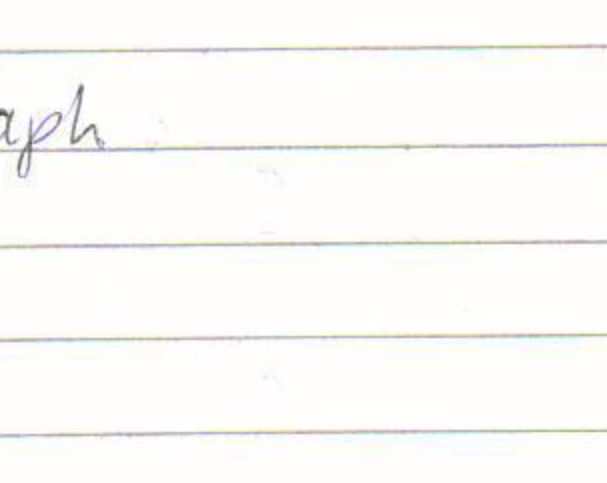


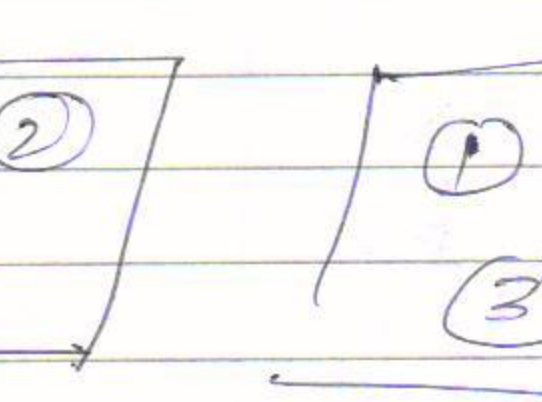


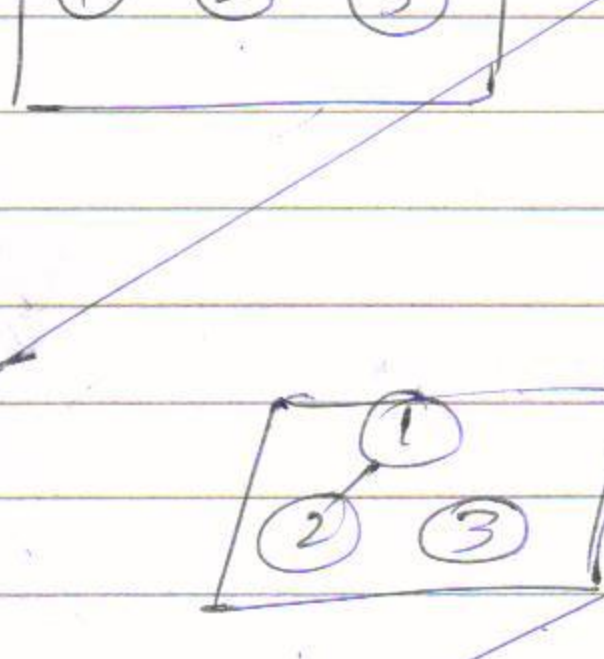












$\frac{3*2}{2} = 3$ edges

Date

with $n = 5$ nodes

$$\frac{n(n-1)}{2} = \frac{5 \times 4}{2} = 10$$

$n = 2 \Rightarrow \frac{n(n-1)}{2} = \frac{2 \times 1}{2} = 1$ edge $\quad 2^1 = 2$

$n = 3 \Rightarrow \frac{3 \times 2}{2} = 3$ edges $\quad 2^3 = 8$ (3 edges) no of simple graph possible

$n = 6 \quad \frac{6 \times 5}{2} = \frac{30}{2} = 15$ (edge possible) $\quad 2^{15} =$ no. of simple graph possible

$n = \frac{n \times (n-1)}{2} = \left[2^{\frac{n(n-1)}{2}}\right]$ Ans.

The no. of simple graph possible $\Rightarrow$

$$\frac{n(n-1)}{2}$$

$$= 2.$$

where n = no. of nodes.

① $\Rightarrow \infty$ Multigraphs possible with n no. of vertices bcoz maximum degree of every vertex is ∞.

7-Nov-12

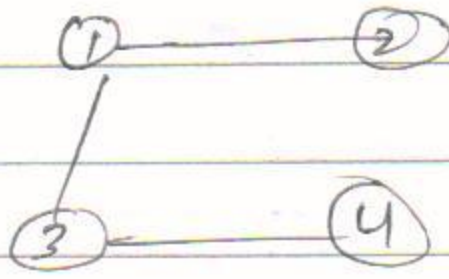
means covering

Date

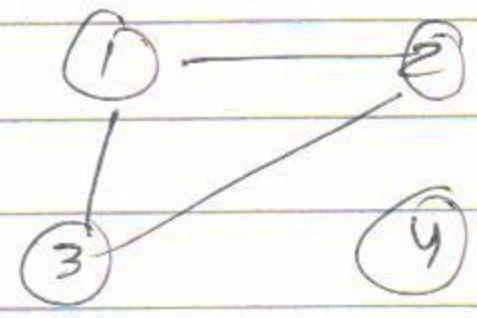
⇒ Spanning Tree :-

To cover any graph ~~min~~ having n vertices, minimum no. of edges required = (n-1) edges. where cycle should ~~be~~ not be there.

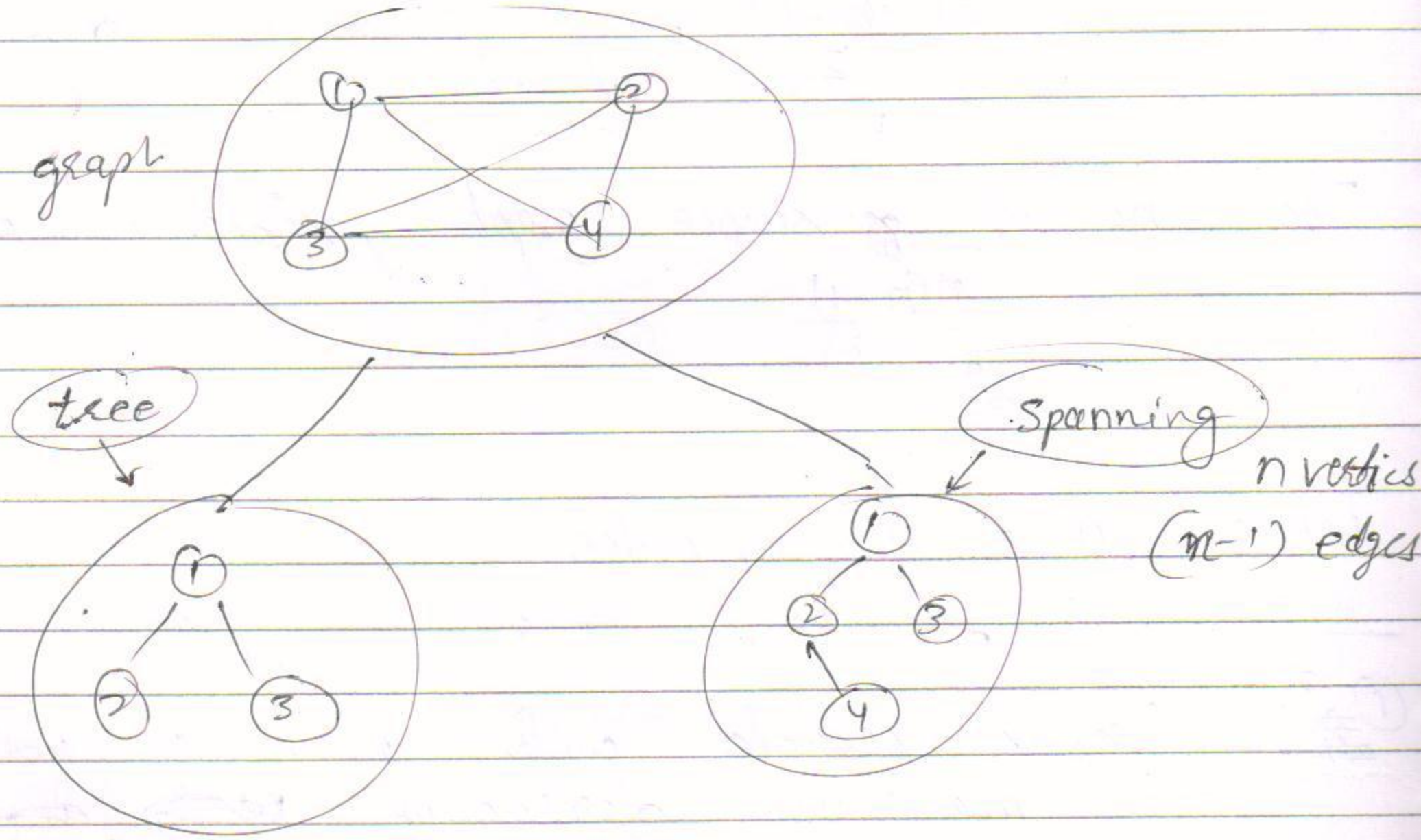
- If n vertics graph is not covered with (n-1) edges. means graph has cycle.



covered by 3 edges



not covered all



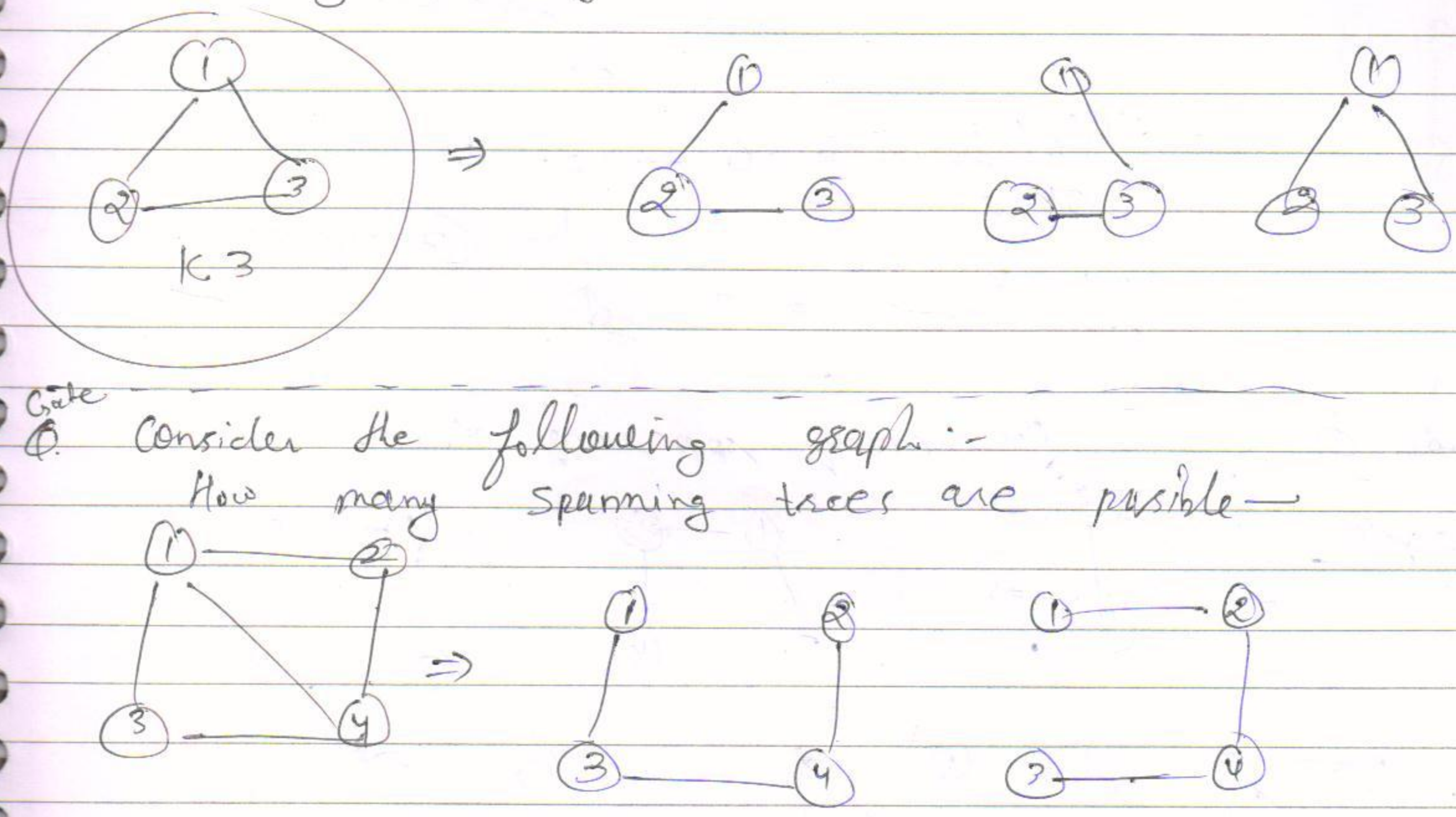
Defination:- A subgraph (S) of a given graph G is said to be Spanning tree iff

① S should contain all vertices of G.
② S should contain (n-1) edges without cycle.

if subgraph is connected then it mean no cycle is there. if contain all vertices and (n-1) edges are there

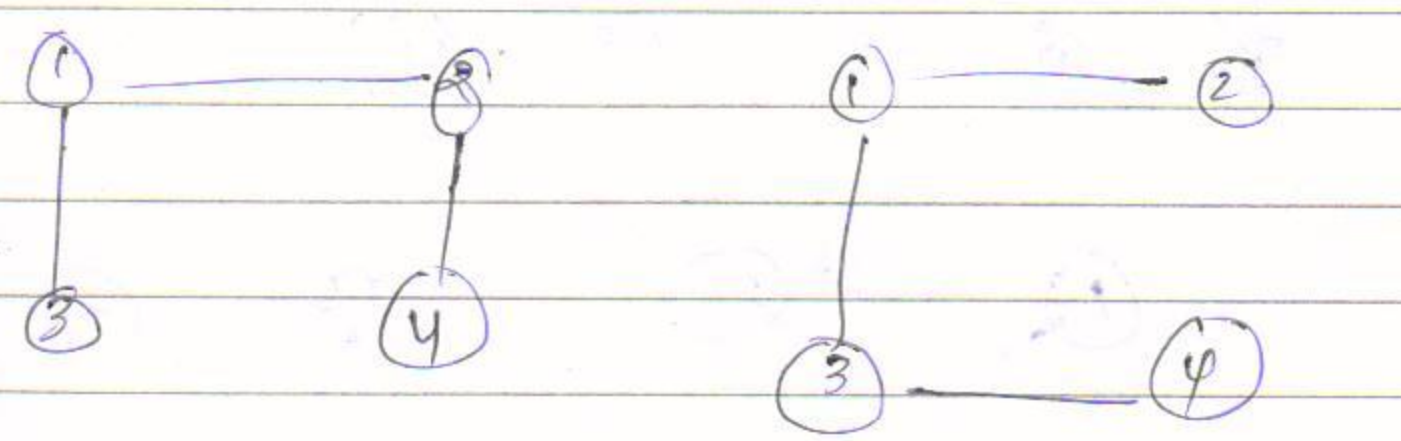
Date

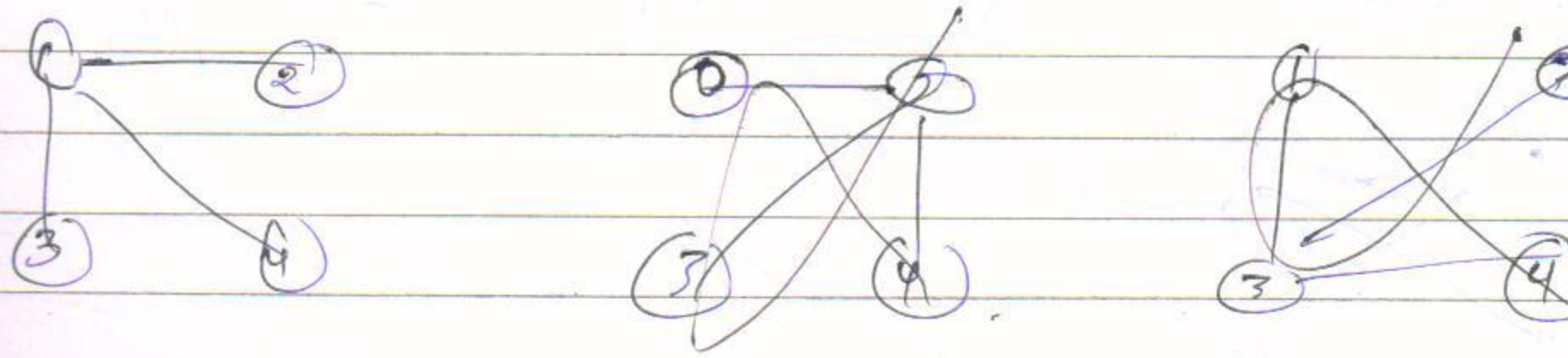
© How many spanning trees are there in

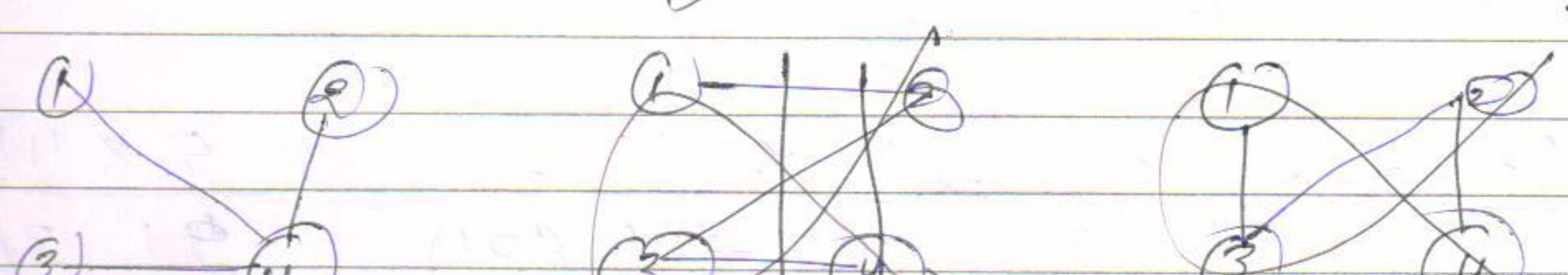


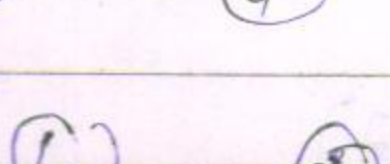
Gate

Q. Consider the following graph:-
How many spanning trees are possible—









Date

(nodes-1) — No of edges

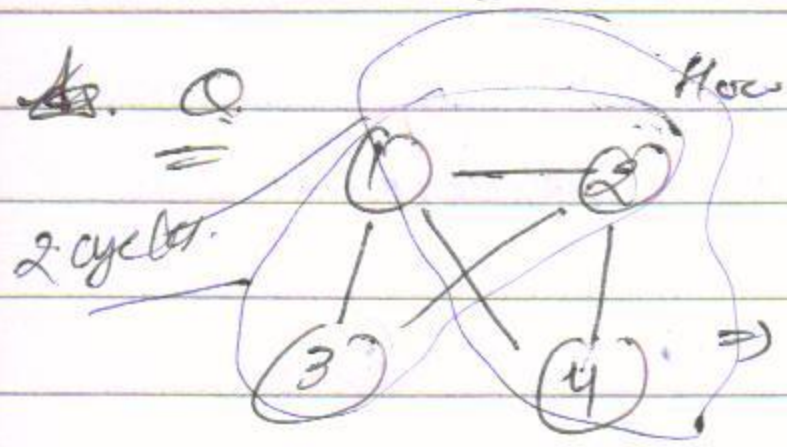
Sol Method $\quad {}^{5}C_{3} = \frac{5!}{3! \times 2!} = \frac{5 \times 4^{2} \times 3 \times 2 \times 1}{3 \times 2 \times 1 \times 2!} = 10$

$= 5$

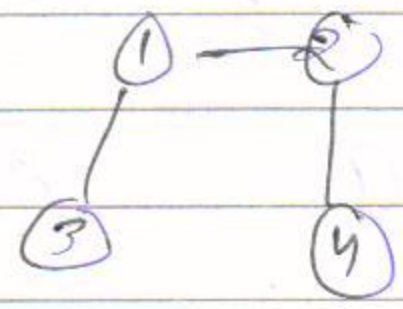
$= \frac{4! \times 5}{3! \times 2!} \Rightarrow 10 - (2)$ — 2 cycles

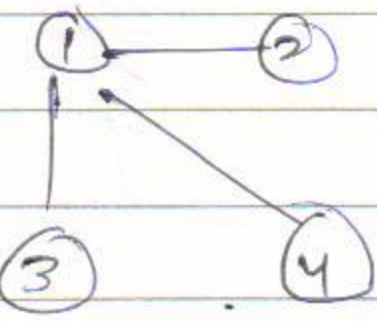
$= \frac{3! \times 4 \times 5}{3!} \Rightarrow 8$ Ans.

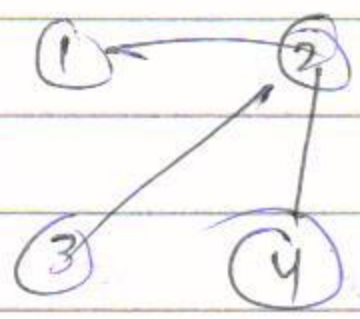
Ans. Q. How many spanning tree.

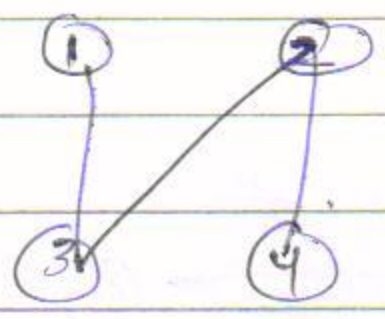


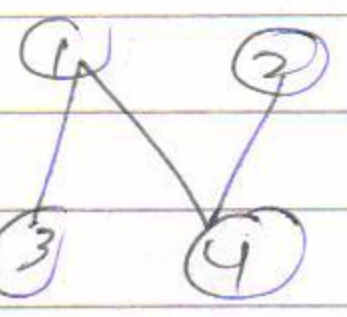
2 cycles

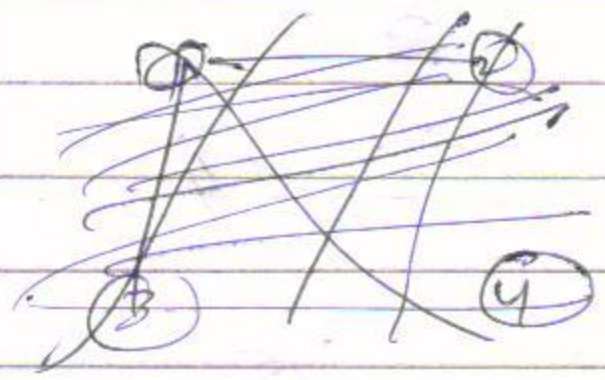


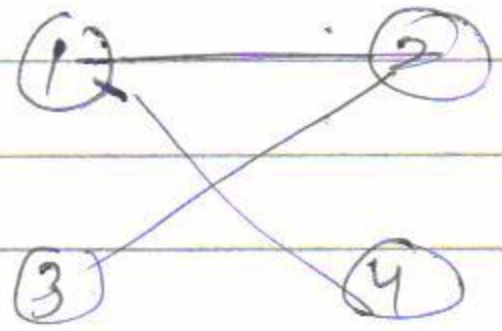


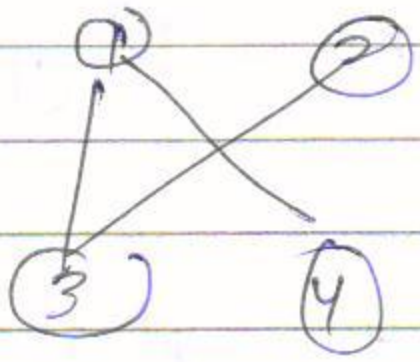


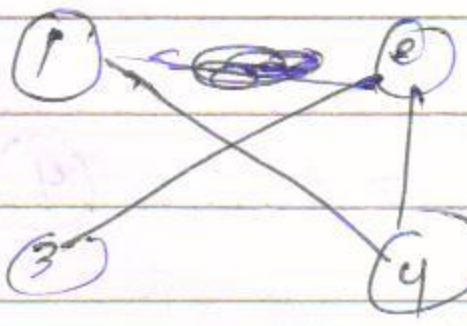












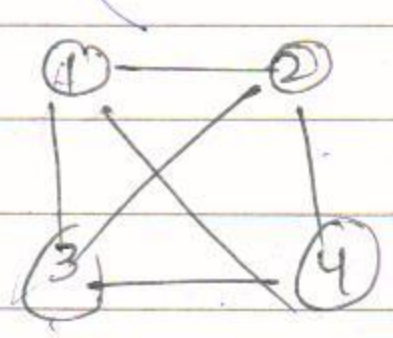
Formula :- $({}^{5}C_{3})$ (Node-1, No. of edges) $= \frac{5!}{(5-3)!\,(3!)} = \frac{5 \times 4!}{2!\,3!}$

${}^{n}C_{r} \Rightarrow \frac{n!}{(n-r)!\,r!}$ $= 10 -$ (2) — 2 cycles.

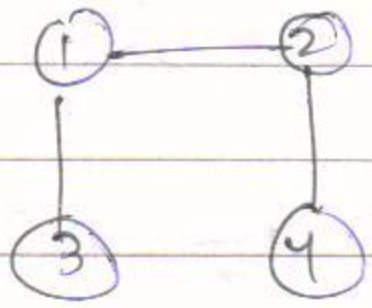
$4-1=3$ → ${}^{6}C_3 \Rightarrow 20-4 \Rightarrow 16.$ (219) $\Rightarrow {}^{6}C_3 = \frac{6!}{3!\,3!} = \frac{6\times5\times4\times3!}{3!\,3\times2\times1} = 20$

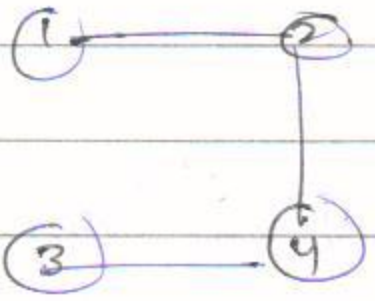
Q. How many spanning ~~graphs~~ Trees are possible.

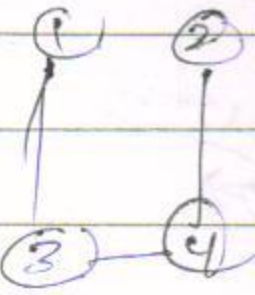
(4^1)



⇒

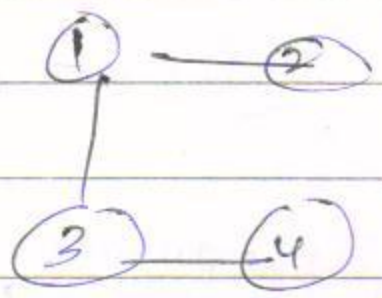


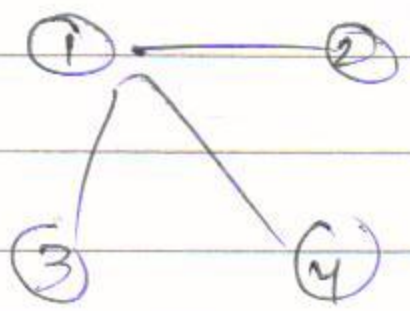


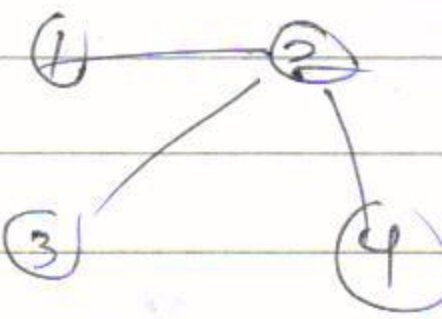


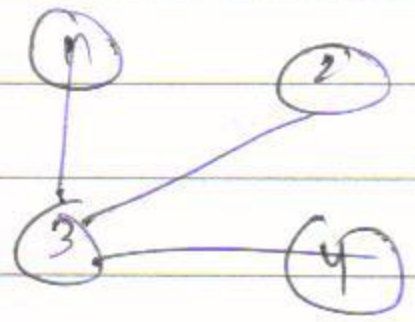
6 eges.

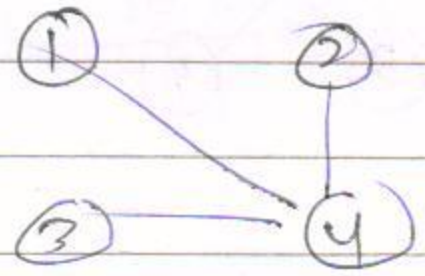
4

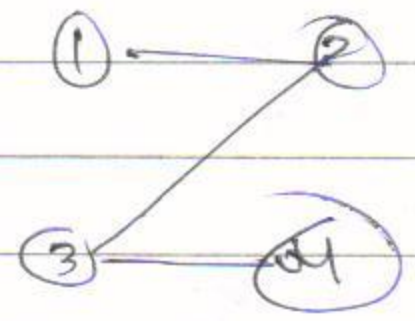


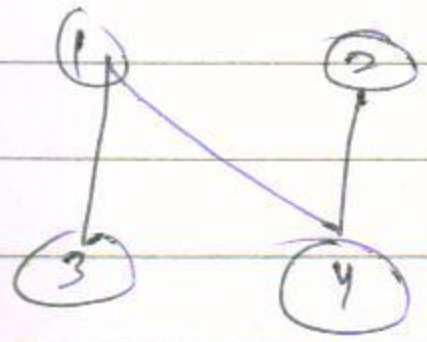


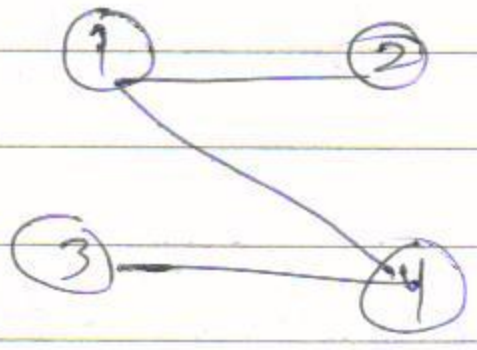


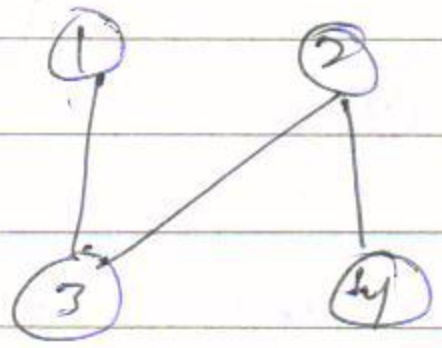


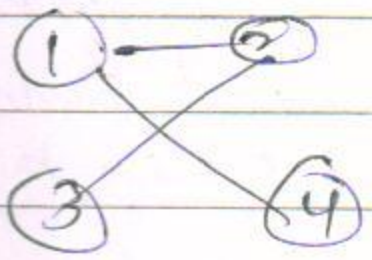


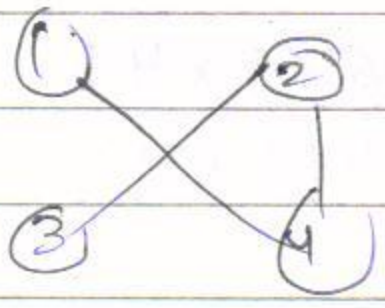


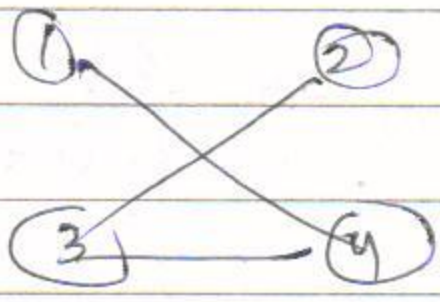


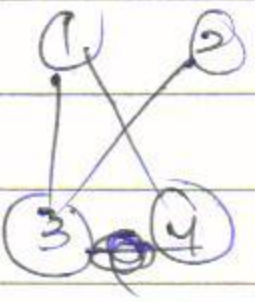












Total = 16 ✓

Complete graph with 3 nodes

$K_3 = 3^{(3-2)} = 3$ ✓

Note :- No of spanning trees

$K_4 = 16 \Rightarrow (4^{4-2}) = 16 \Rightarrow 4^{4-2} = 4^2 = 16$ ✓

complete graph with 4 nodes.

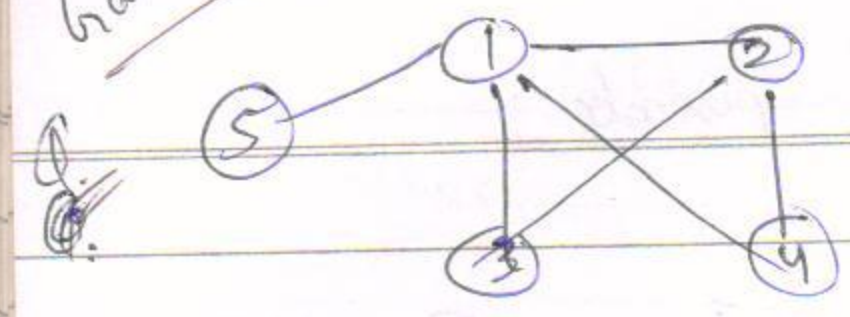
$K_5 = 125 \Rightarrow (5^{5-2}) = 125 = 5^{5-2} = 5^3 = 125$ ✓

$K_n \Rightarrow n^{n-2}$

Gate Ques

$^6C_4 \Rightarrow 15 - 7 = 8$

220

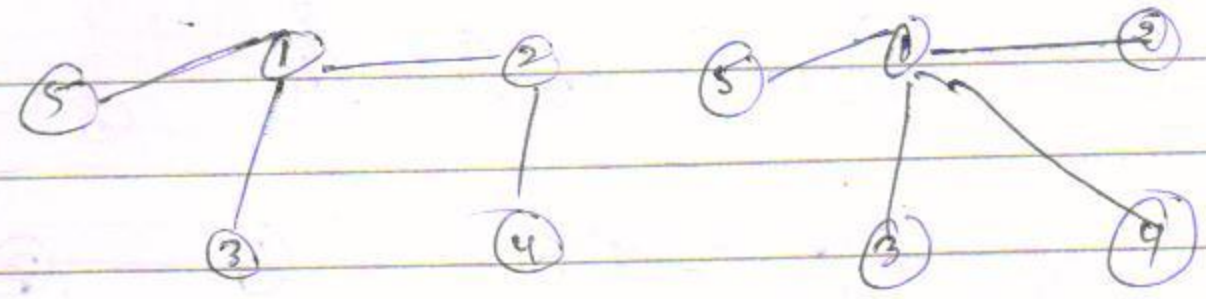


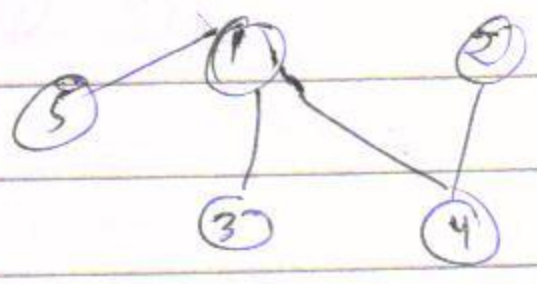
Q. ⇒ How many spanning trees are possible.

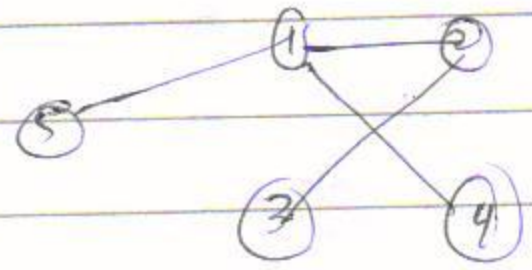
Ans Total node = n = 5
Spanning tree contains edges = (n-1) = 5-1 = 4

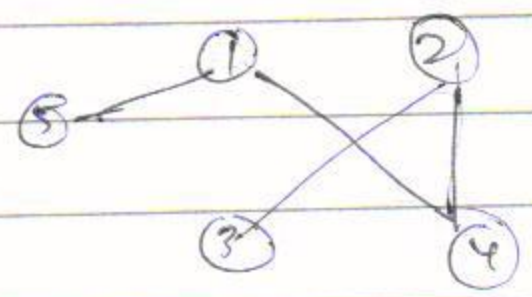
Ans

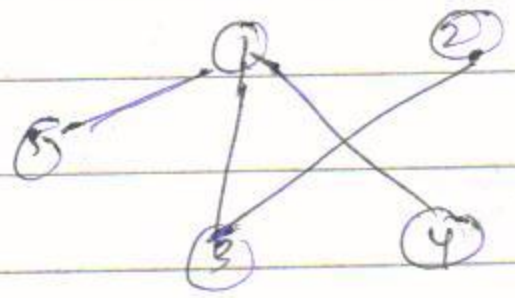
⇒

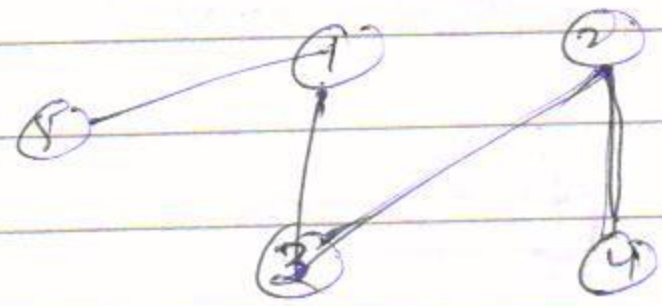






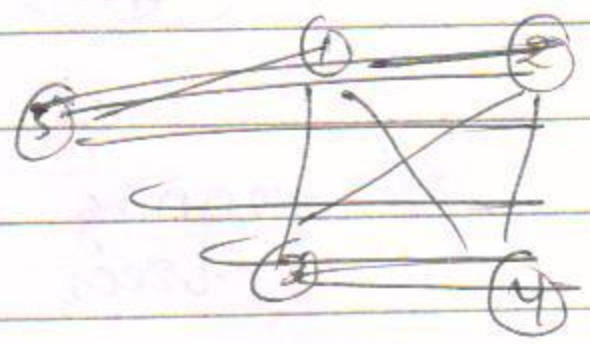


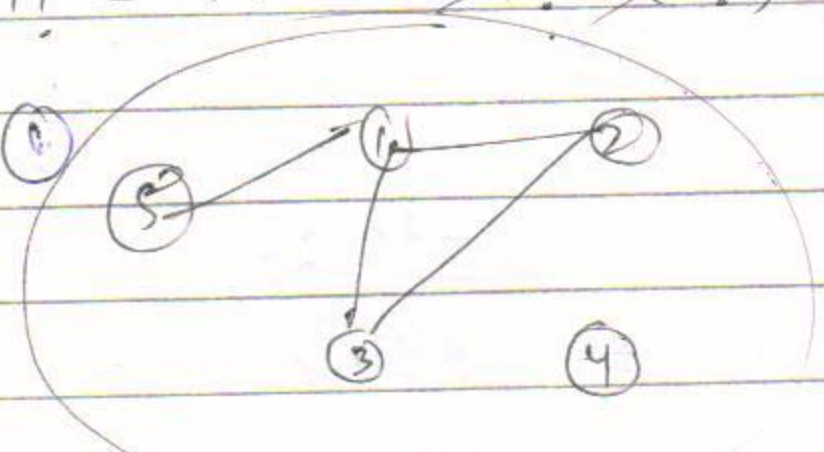


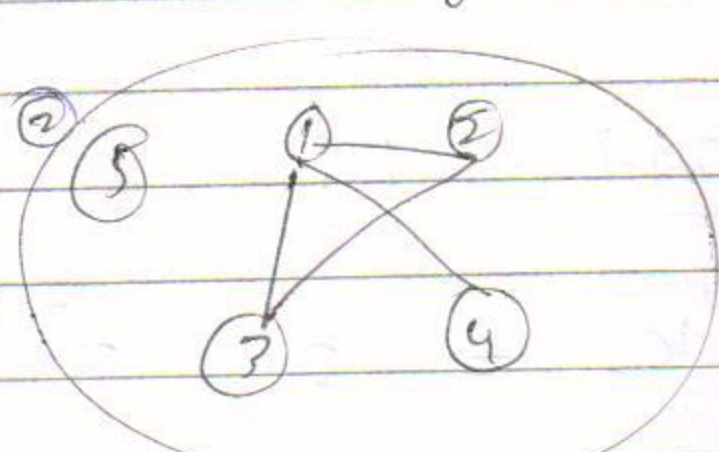


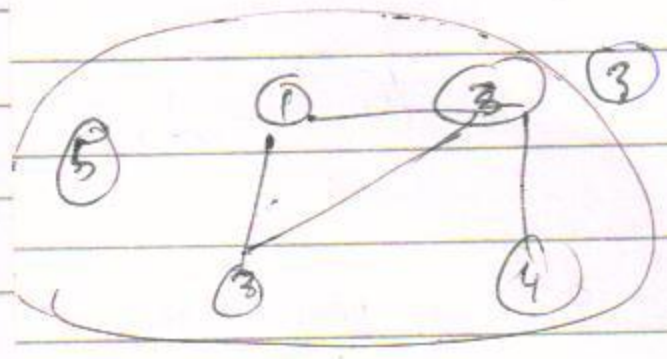
total 8 spanning trees will be there.

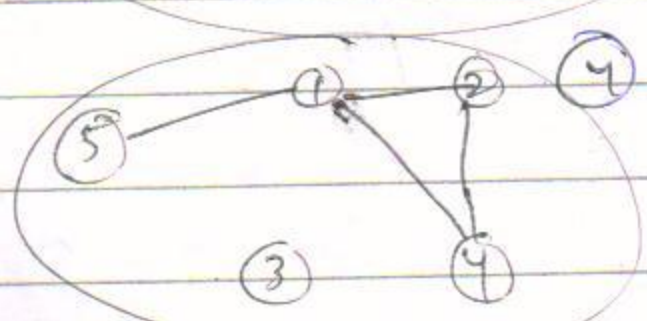
Check:- $^6C_4 \Rightarrow \frac{6!}{4!(6-4)!} = \frac{^3\cancel{6} \times 5 \times \cancel{4!}}{\cancel{4!}\,(\cancel{2!})} = 15$ — (all possible graph having cycle with 4 edges)

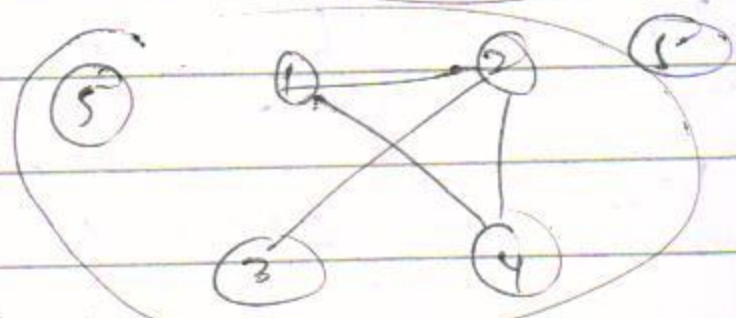


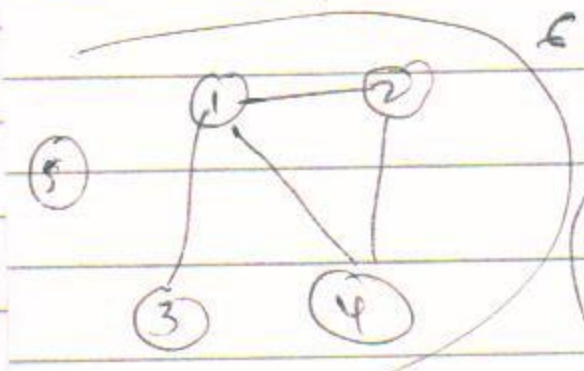


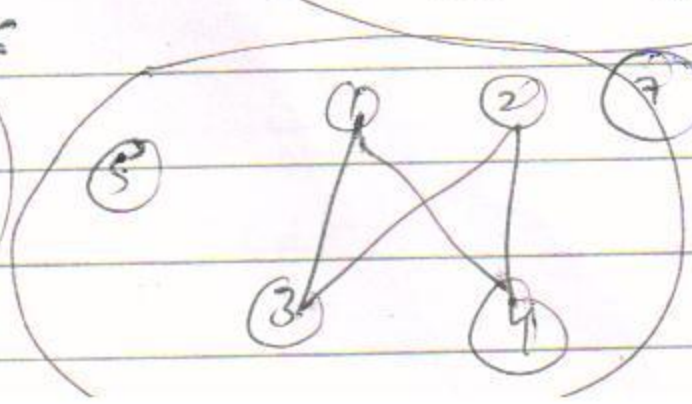












⇒ 15 - 7

= 8 spanning tree.

Use → Constructing minimum cost ₹ computer network.

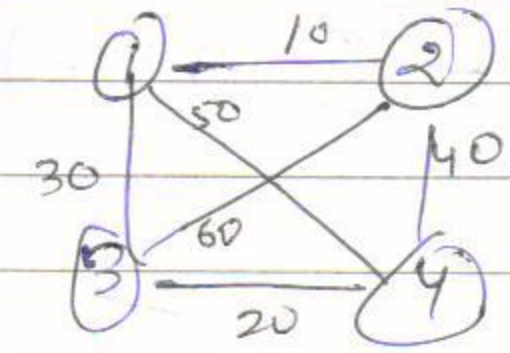
221

Date

⇒ Networking people – use spanning tree.

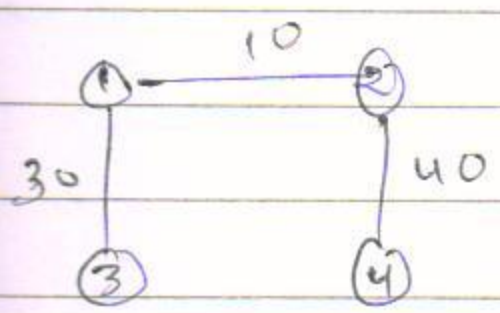
⇒ Minimum Cost spanning tree :–

← weighted graph.

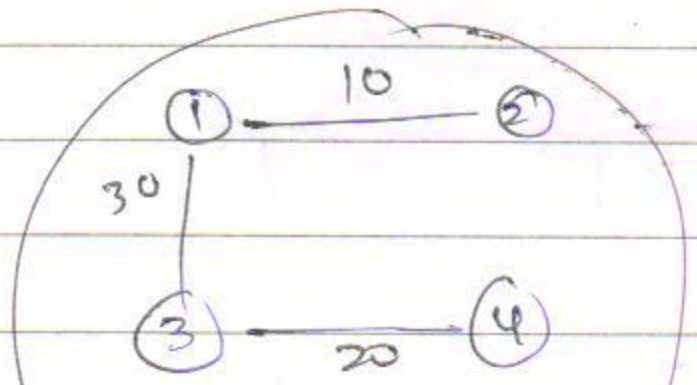


all minimum 3 edges are there so this is minimum.

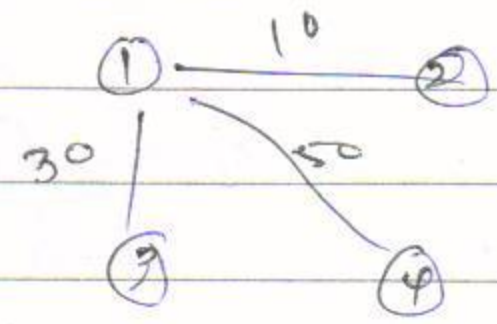
↳ draw spanning tree



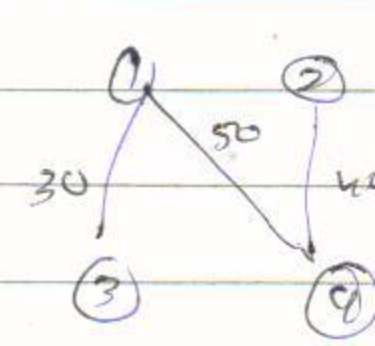
50



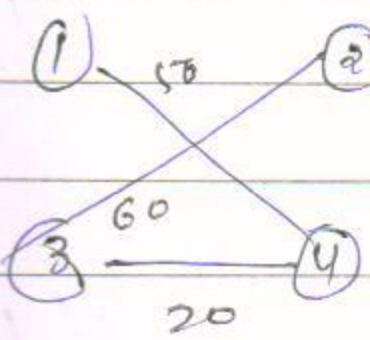
60



90



120



130

Minimum spanning tree.

To find minimum spanning tree there are 2 Algos–

(1) Kruskal.

(2) Prims

G (V, E).

↳ No of edges $|E| \leq \frac{V(V-1)}{2}$

eg. G (4, E)

$E \leq \frac{4 \times 3}{2}$

$|E| \leq 6$

So $E = O(V^2)$

$\log E = O(2 \log V)$

Date

$n^2 = o(n^3)$

$2\log n = o(3\log n)$

$\log n = o(\log n)$

$n! = o(n^n)$

$\Downarrow$

$\log(n!) = \Theta(n \log n)$

$a = o(b)$
└ small

$\log a \neq o(\log b)$

#Note:- If $a = O(b)$ then $\log a = O(\log b)$ — alway true in the case of big-O.

$\Rightarrow 2^n = O(2^{2n})$

$n \log 2 = O(2n \log 2)$

$n = \Theta(n)$

} sometimes it will be equal.

$\Rightarrow \log n = O(n)$

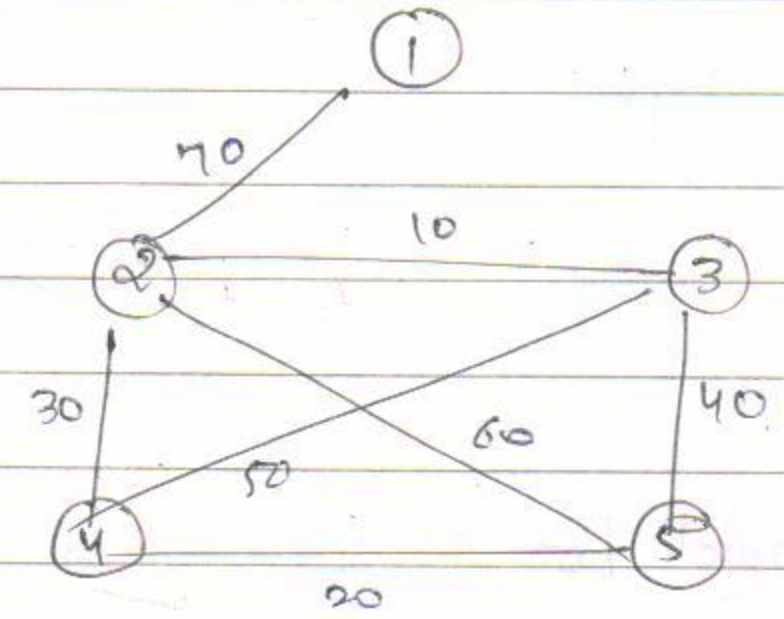
$\log(\log n) = O(\log n)$

} Obig-O same cases

$\Rightarrow$ if $a = o(b)$
└ small o
then
$\log a \neq o(\log b)$

Date

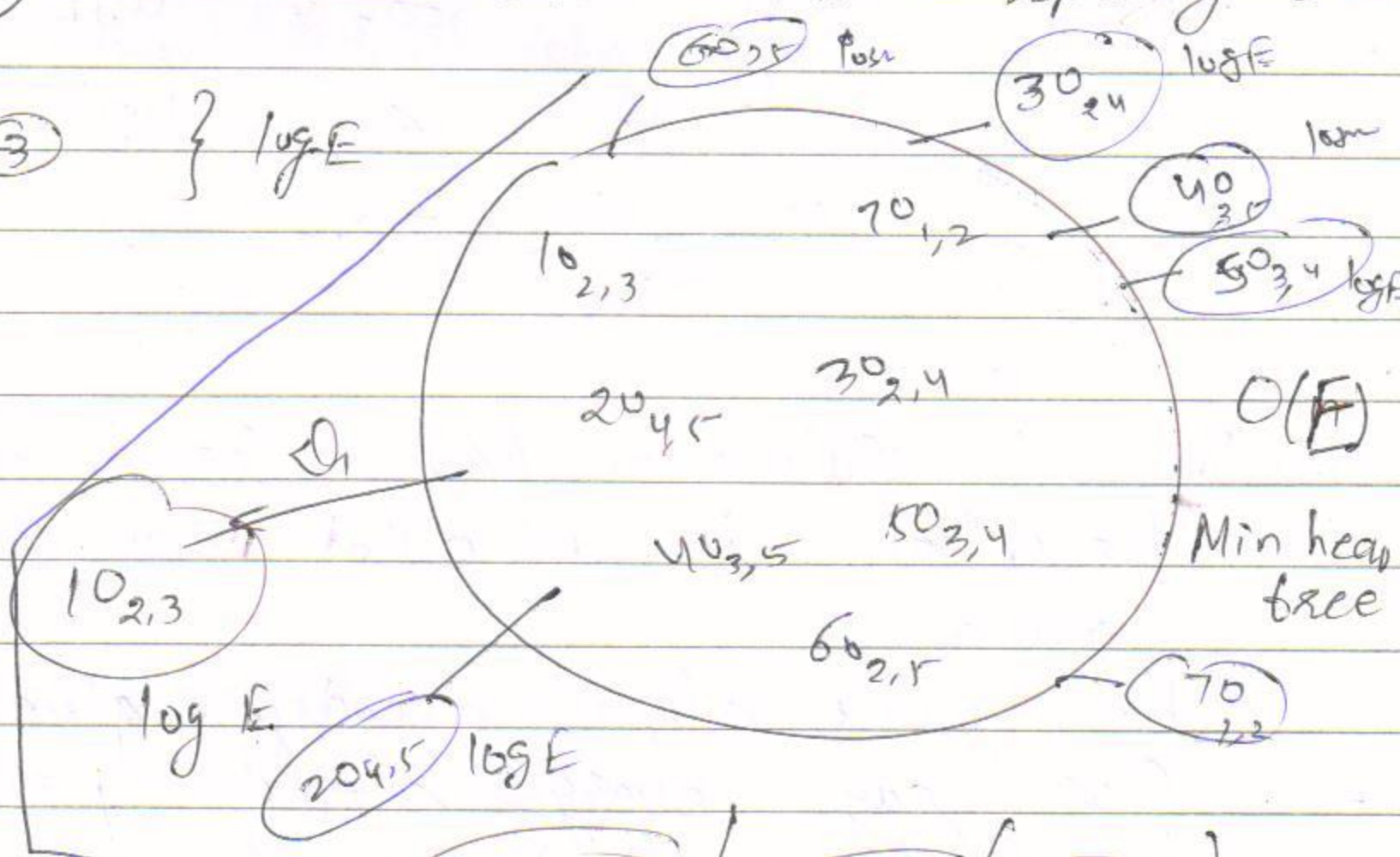
⇒ Kruskal Algo :-



Steps :-

① Take 1 minimum edge - weight (Min heap tree will give after deletion) and adel to minimum spanning tree.

(2) —10— (3) } log E

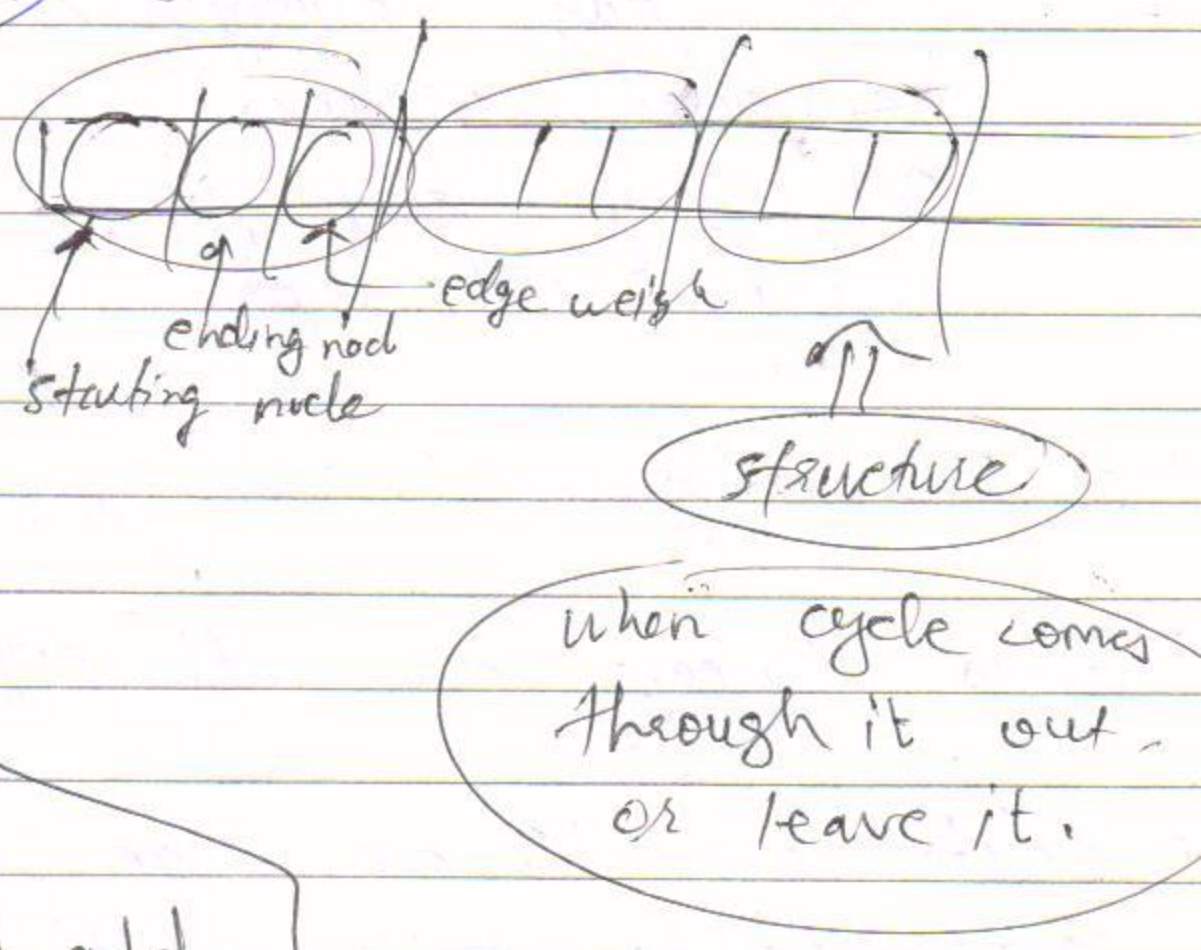


② Take next Minimum edge and add to MST.

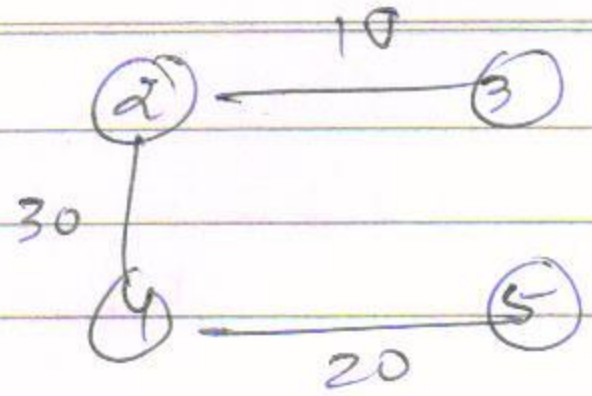
(2) —16— (3)

(4) —20— (5) } log E

③ take next-min-edge and add to MST if cycle is not formed.

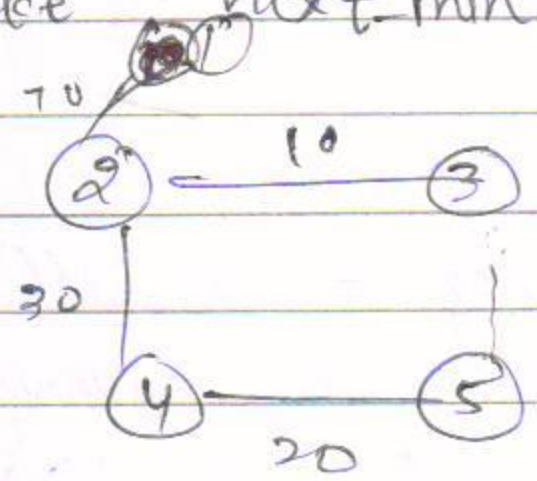


Date



} logE

④ Take next min-edge and add to MST if no-cycle



cycle coming so leave it

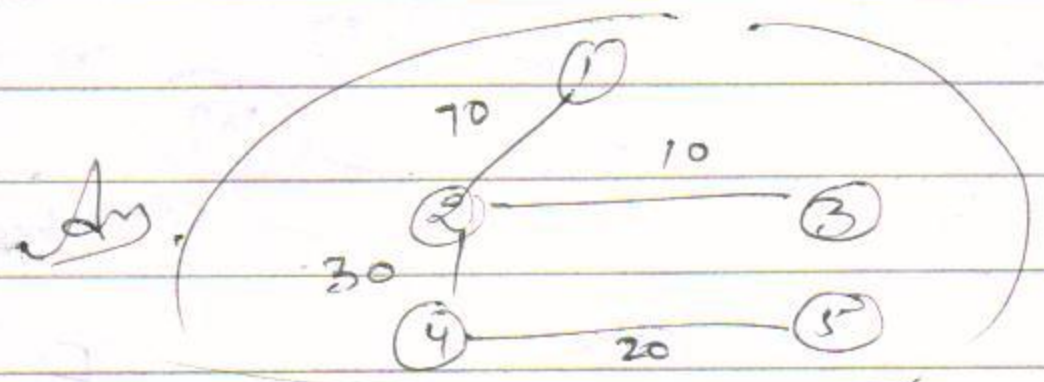
$40_{3,5}$ — logE time wasted

$50_{3,4}$ logE

$60_{2,5}$ logE

$70_{1,2}$ logE

Ans.



Fundamental Difference b/w Krushkal & prims.

→ Krushkal in middle may get disconnected graph

but prims algo always give connected graph.

←→ (or say every step of prims algo generate connected graph.)

But any step of krushkal Algo may generate disconnected. graph

Kruskal - Algo .T.C :—

	B.C.	W.C.
① Create minheap for all edge-weight.	O(E)	O(E)
② Delete one by one edge from min heap and add to MST until (V-1) edges added.	(V-1) logE Up E+V logE E+V log V	E log E Up

	B C	W C
(1)	O (E)	O (E)
(2)	(V-1) log E ⇓ E + V log E ⇓ O (E + V log V)	E log E ⇓ E + E log E ⇓ E + E log V ⇓ O (E log V) ↳ always. ↳ average case.

depending on the graph

if normal graph then it will take (V-1) edges so V log V is greater.

&& if complete graph is there max edges ⇒ E^2

so (E^2) + V log V

big.

And that is depend on given graph so take as it is and according to graph decide. T.C. of best case.

---

* if we go for array besides minheap.

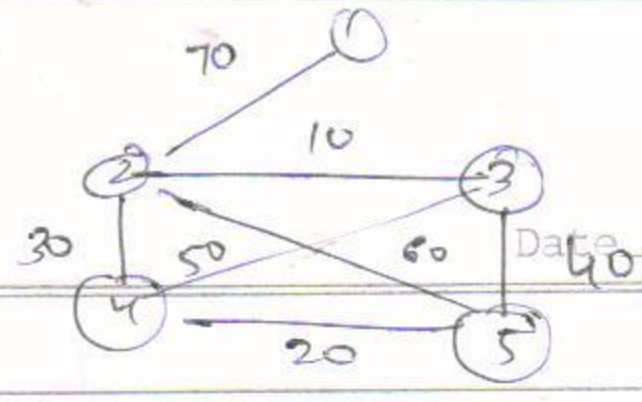
to sort array of E edges.

B.C. ⇒ E log E + V ⇒ E log E

taking (selecting) (V-1) elements (egdes) to create (MST)

W.C ⇒ E log E + E ⇒ E log E

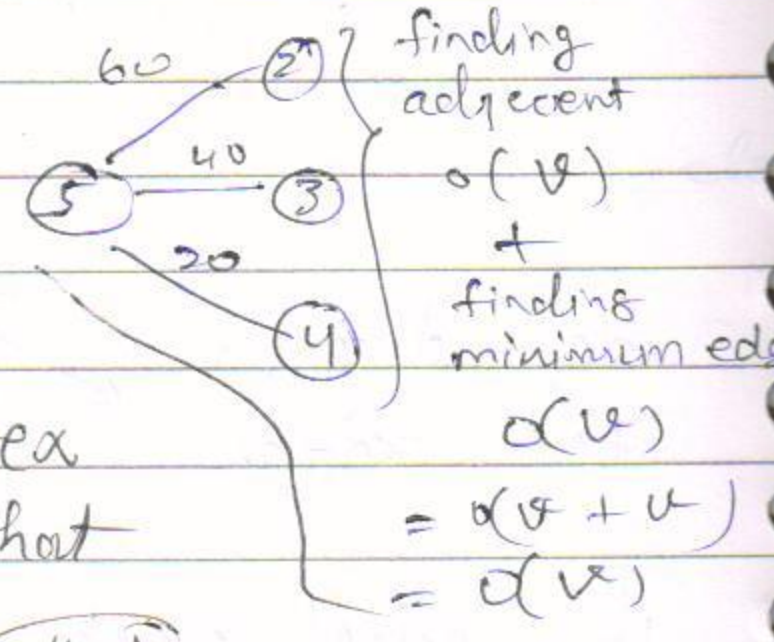
5. Minimum spanning tree :-



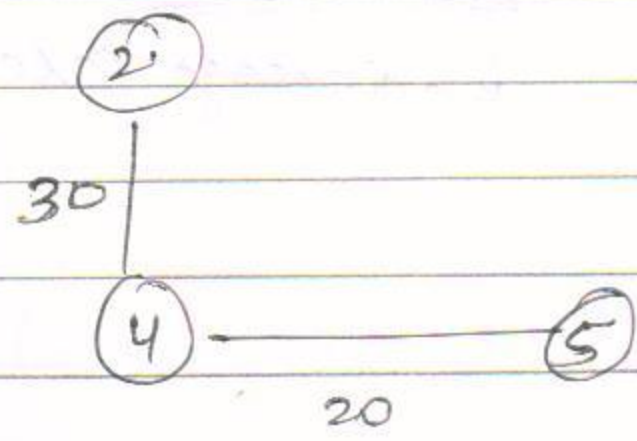
⇒ Prims Algo :-

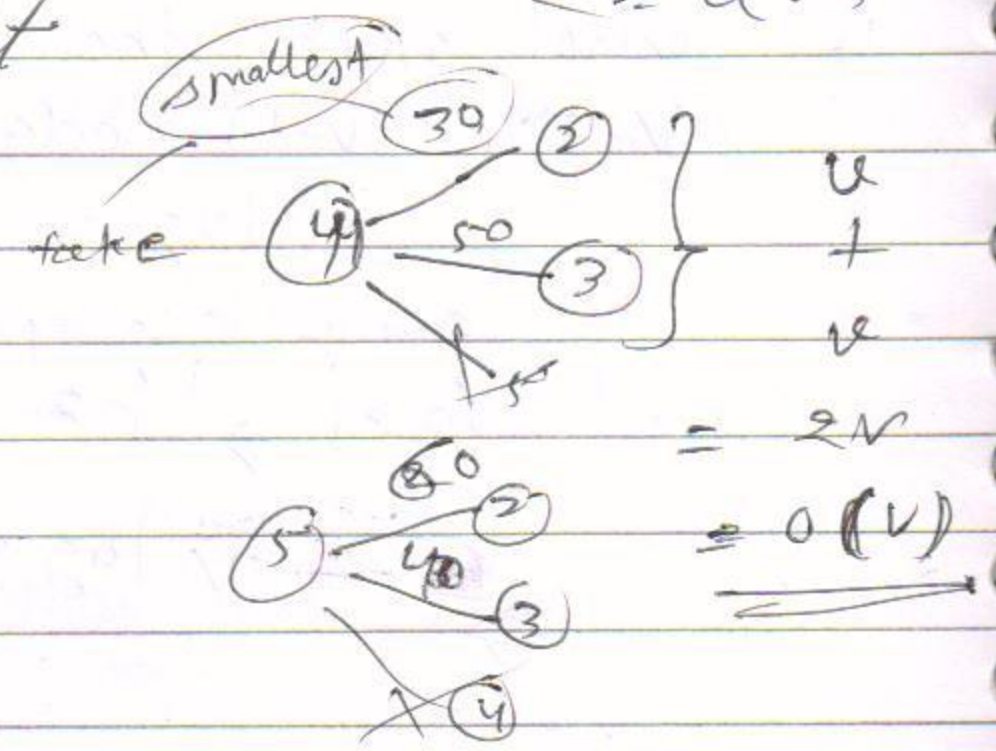
Steps :-

① Take any vertex and find adjacent of that vertex and take minimum cost edge among them :-

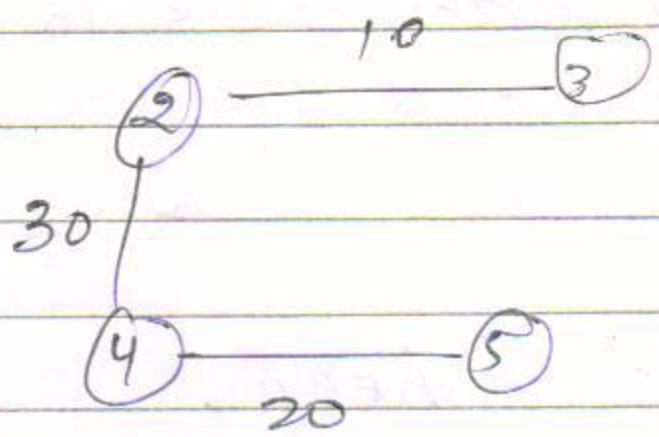


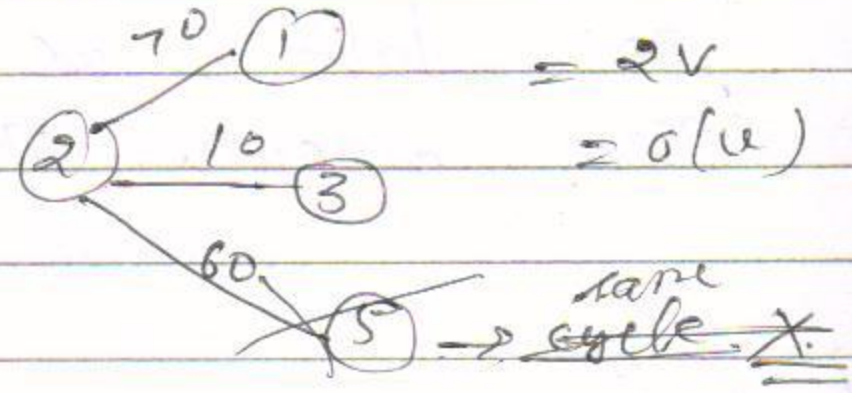
② Take new vertex or find new vertex adjacent and take minimum of of that adjcent and previous adjcent



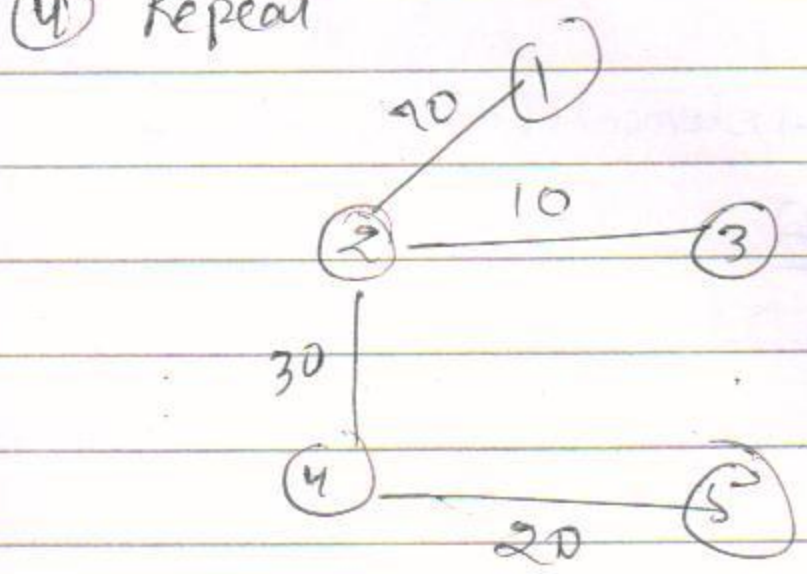


③ Repeat step means -





④ Repeat



all are already over both.

3

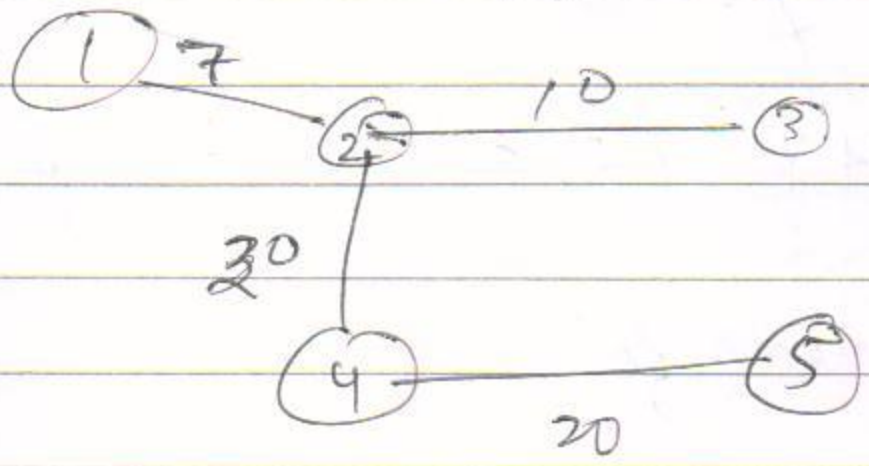
other are making cycle so leave

Only

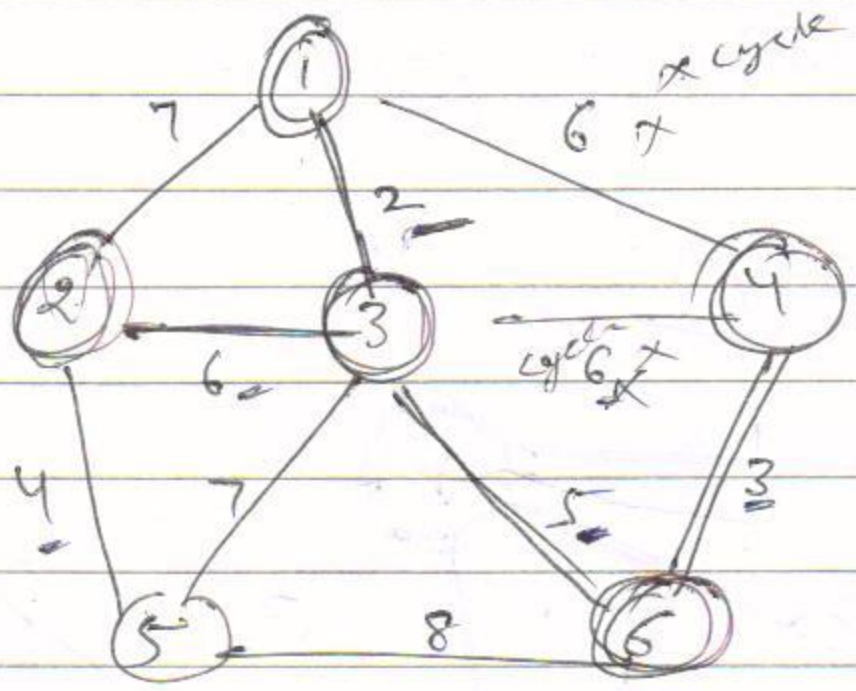
Date

T.C ⇒ $\underbrace{2V + 2V + 2V + 2V \text{ ----- } + 2V}_{V \text{ times}}$

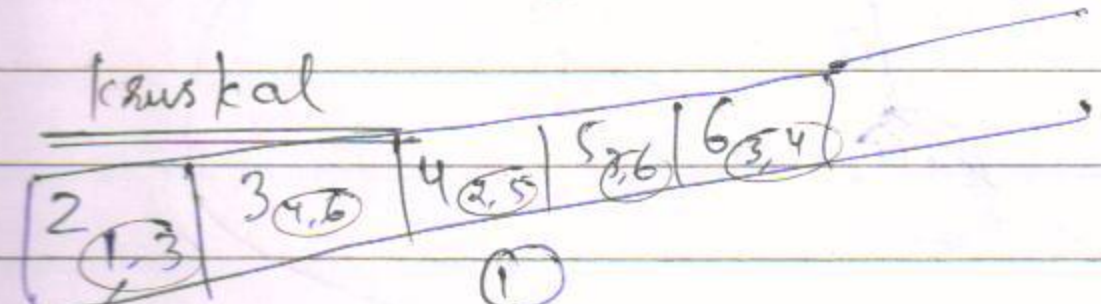
$= O(V^2)$ { using array or adjacency matrix }



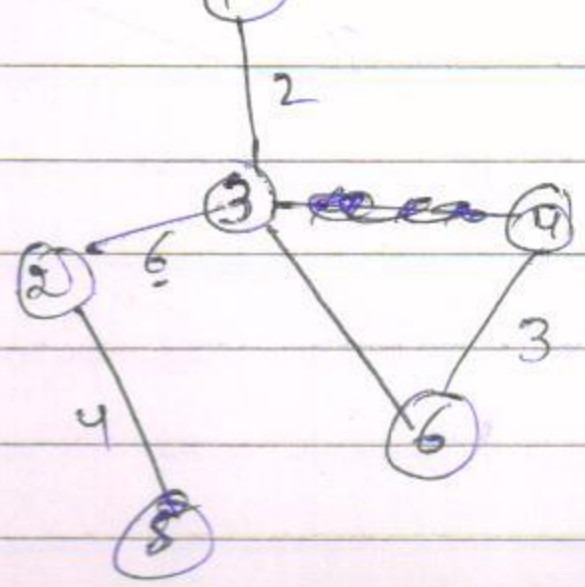
Q. Consider the following graph:-

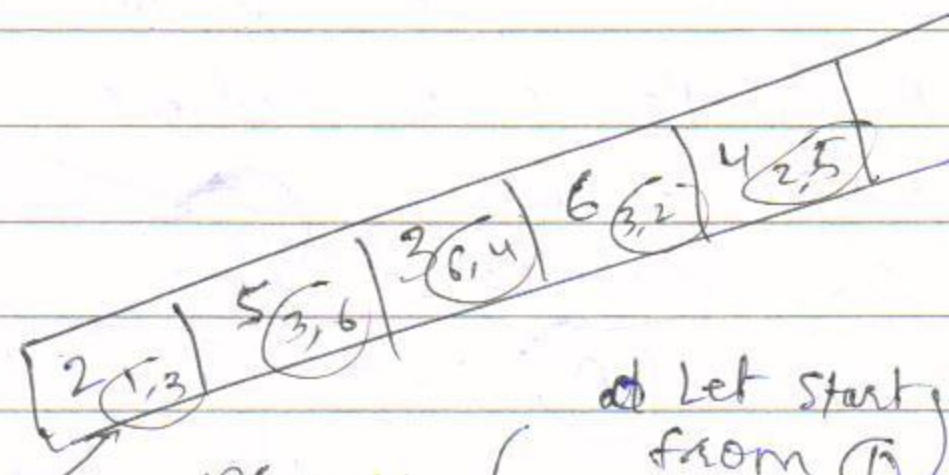


Kruskal



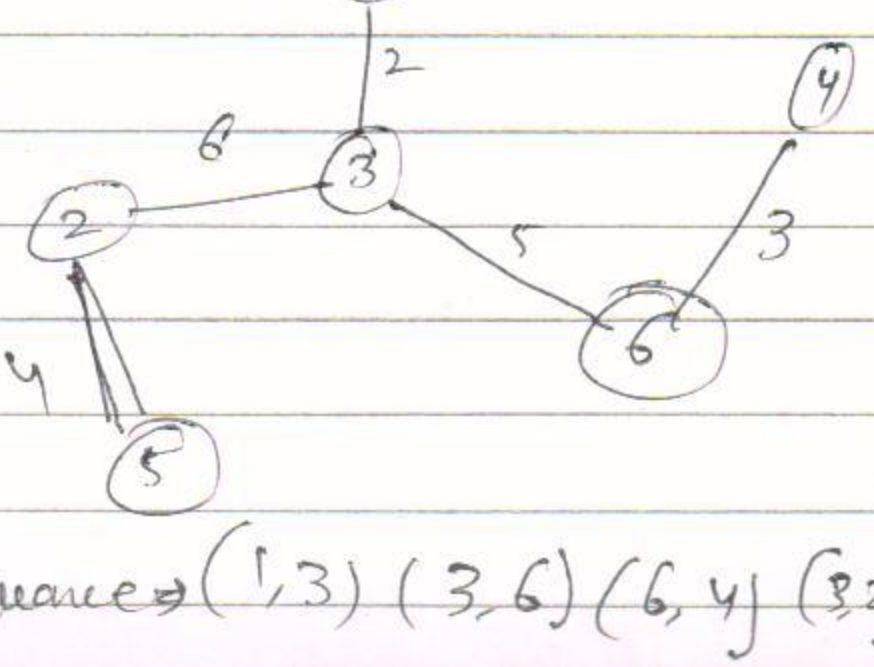
sequence





sequence.

Prims. (Let start from (1))



Sequence ⇒ (1,3) (3,6) (6,4) (3,2) (2,5)

Q.

Gate Question come like this on sequence of Date
kruskal & prims

adjacency property → adjacency property satisfy.

adjacency property

(a) (1,3) (3,6) (6,4) (3,2) (2,5)

(b) (1,3) (3,6) (2,5) (3,2) (6,4) adjacency

(c) (1,3) (3,6) (2,5) (6,4) (

(d) (1,3) (3,6) (3,2) (6,4) (2,5) adjacency

⇓

minimum fails.

Ans. (a).

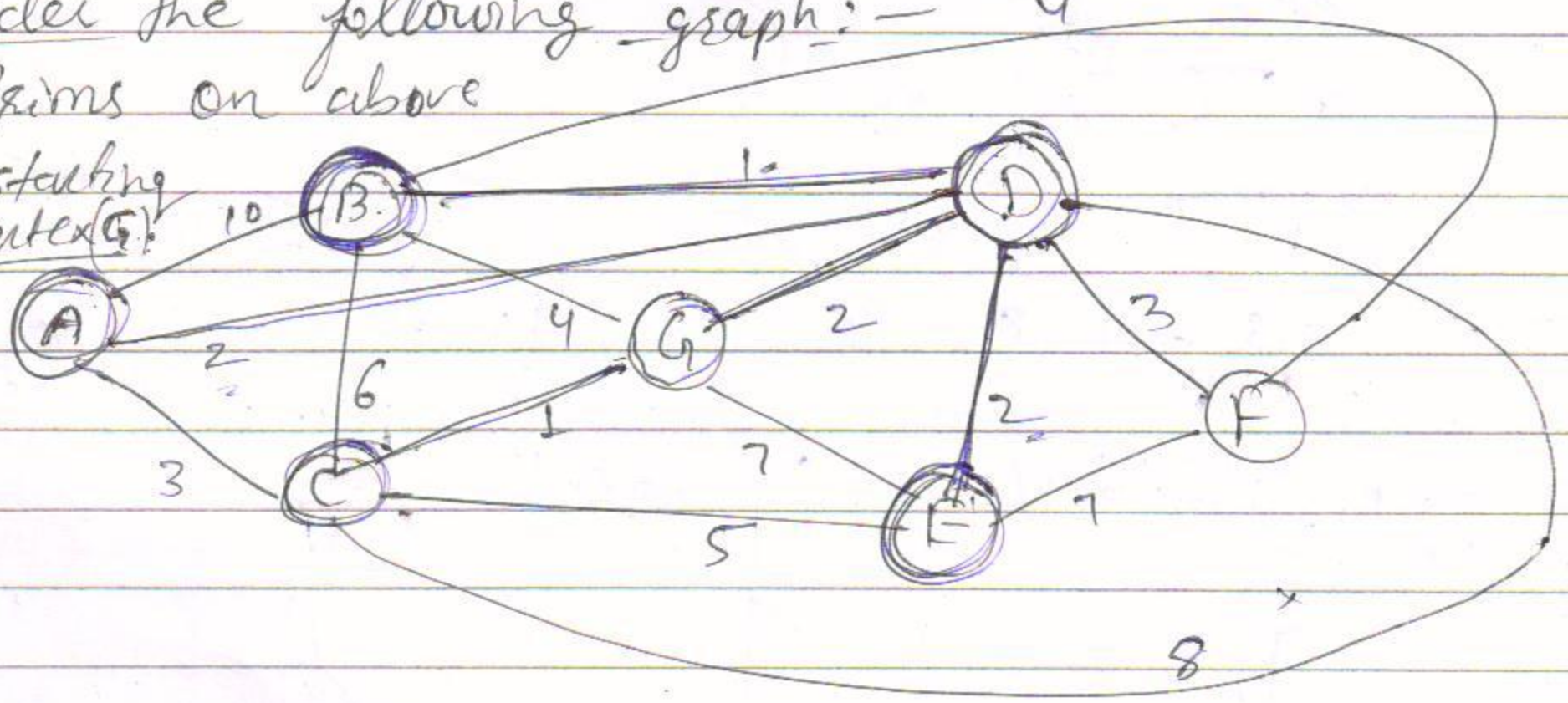
⇒ Method :-

① First try for adjacency property

② Second try for Minimum property.

Gate

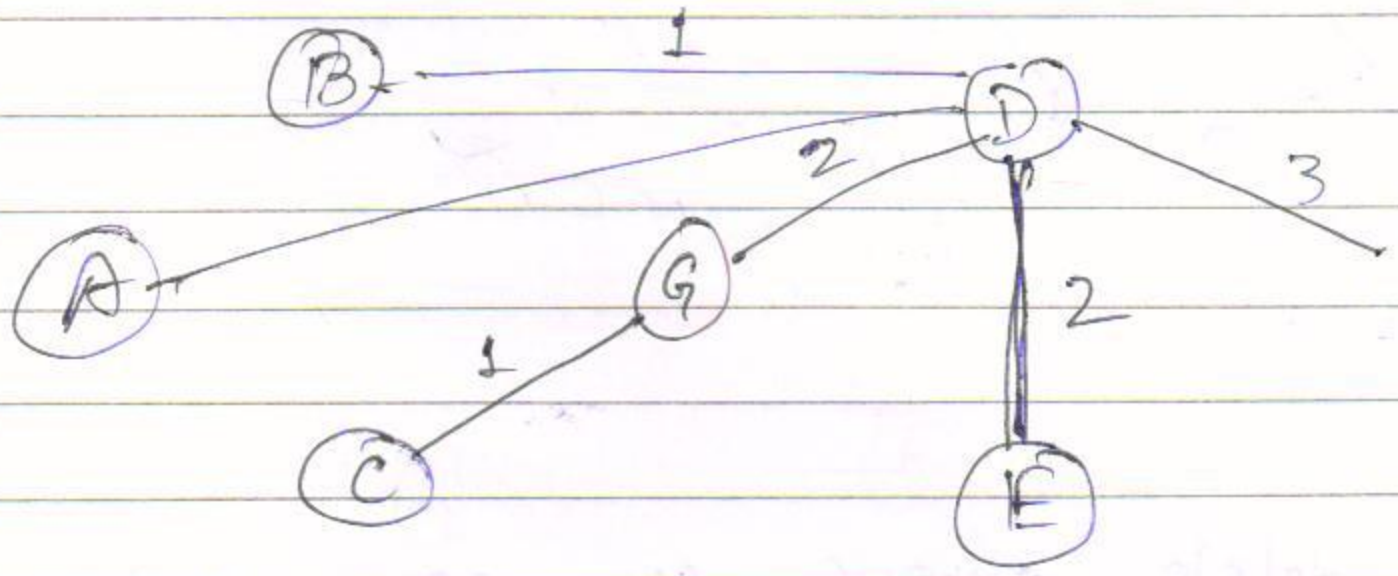
Q. Consider the following graph :- Apply Prims on above graph. starting from vertex (G).



Date

No of vertex = 7

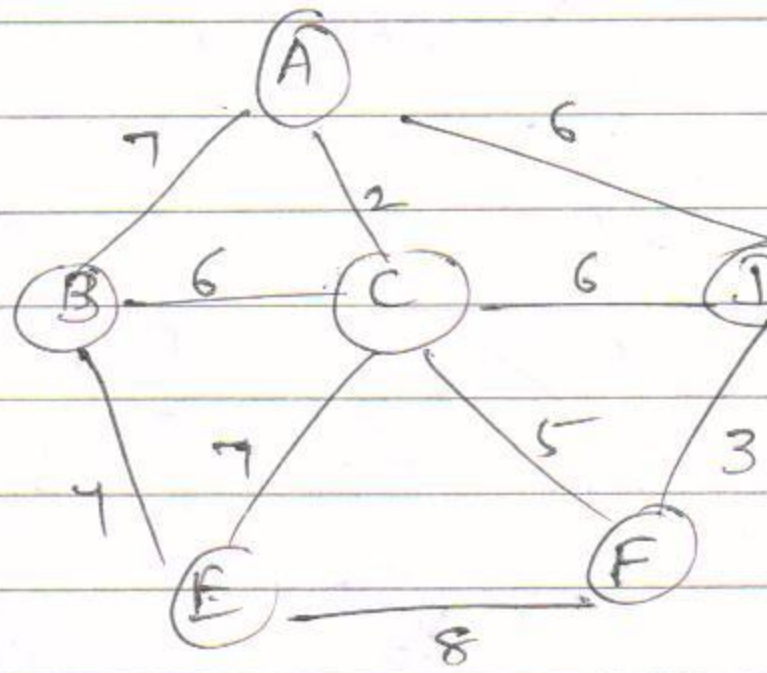
Start from G. ⇒ | 1 G C | 2 G D | 1 D B |



(G C) (G, D) (D B) (D, E) (D, A) (D F)

(G C) (G D) (D B) (D, A) (D, E) (D F)

⇒ Efficient way of implementing Prims Algo :- Using Min heap



Date

Completed

A. min = ∞ 0
A. parent = N

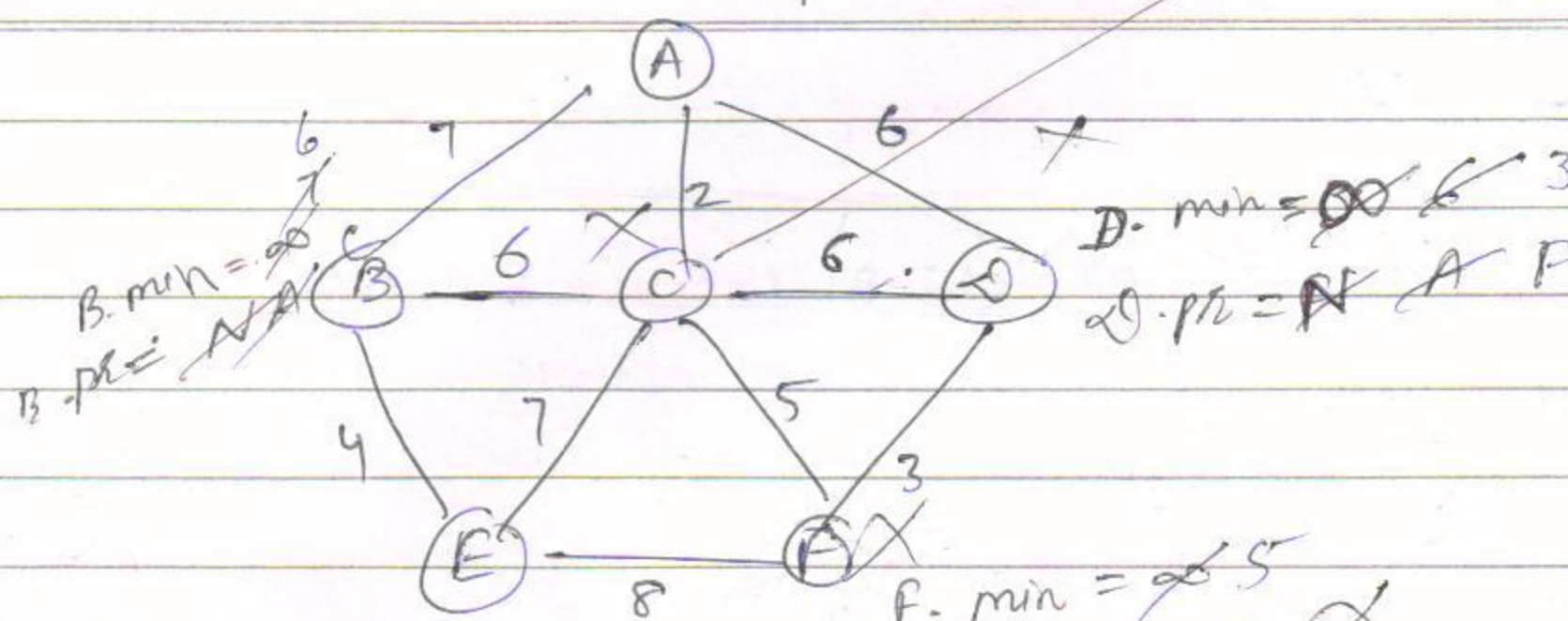
C. min = ∞ 2
C. pr = A

B. min = ∞
B. pr = A

D. min = ∞ 6 3
D. pr = N A F

E. min = ∞ 7 4
E pr = N C B

F. min = ∞ 5
F. pr = N C



- Whenever we delete anyone can come because all are ∞ initially.

TC = O(V) + O(V² log V)
RHS.

V log V — V times deletion.

create minheap O(V)

Not in Minheap	Present in Minheap					
	A	B	C	D	E	F
	∞	∞	∞	∞	∞	∞
	N	N	N	N	N	N

- if start from A make it zero,

	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
	N					
		B	C	D	E	F
		7	2	6	∞	∞
		A	A	A	N	N
C		B	C	D	E	F
		6	6	7	5	
		C	A	C	C	
F		6	3	7		
		C	F	C		
D			6	7		
			C	C		
B						

log V — changing in min heap — best Adjacent A

V log V

C — best Adjacent → AC

V log V

No change

F — Adjacent AC F

log V

D — Adjacent AC F

log V

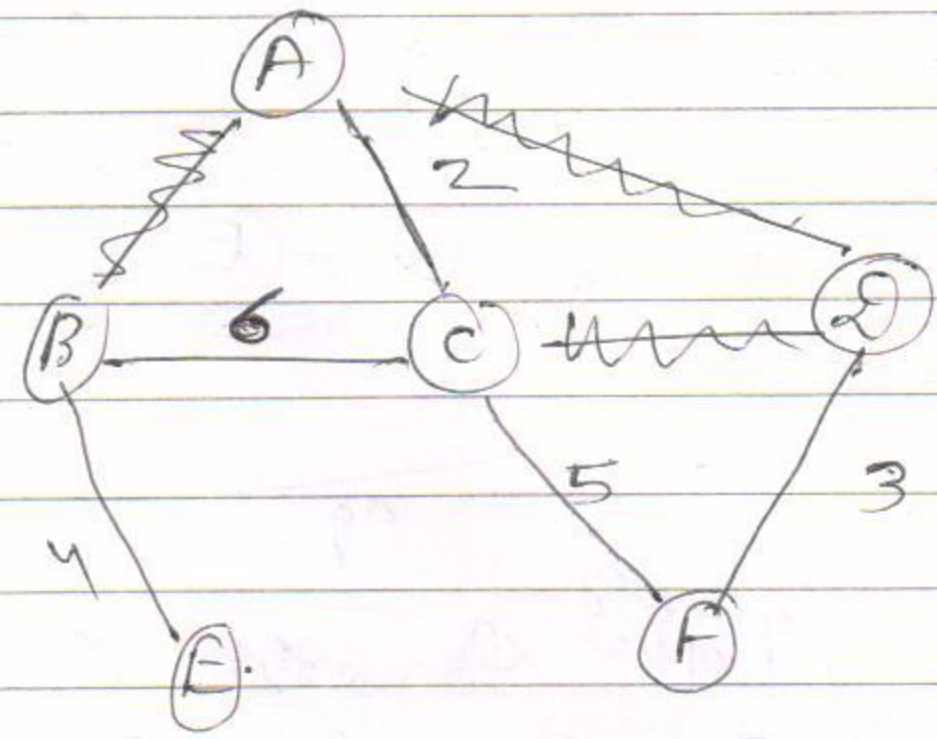
B

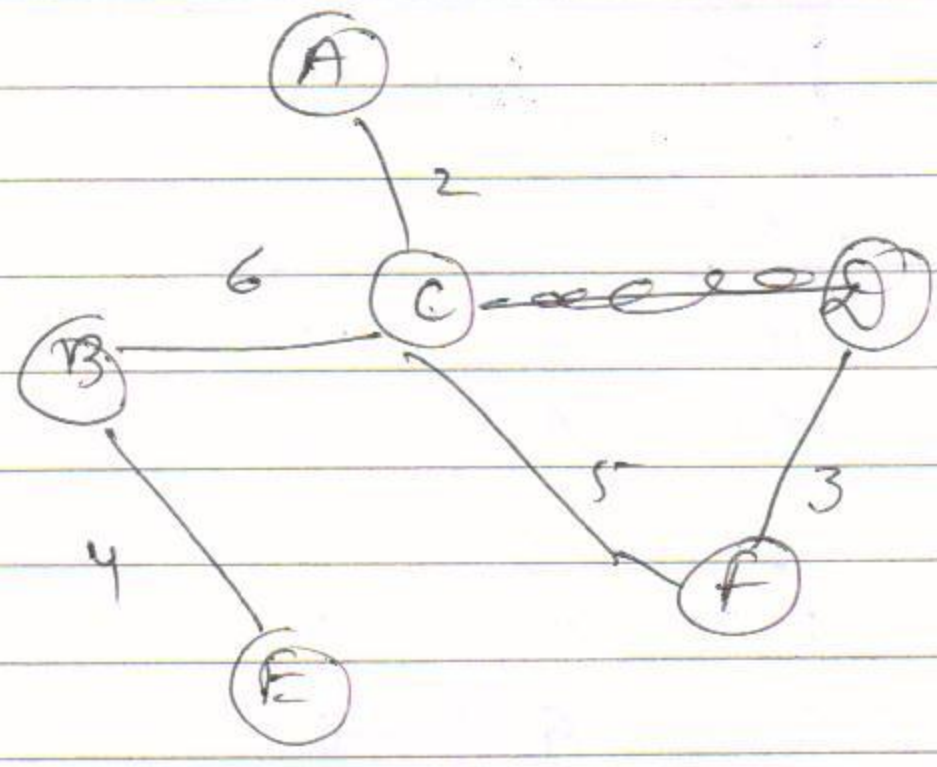
log V

Adj

Date

This table is nothing but Min heap tree.

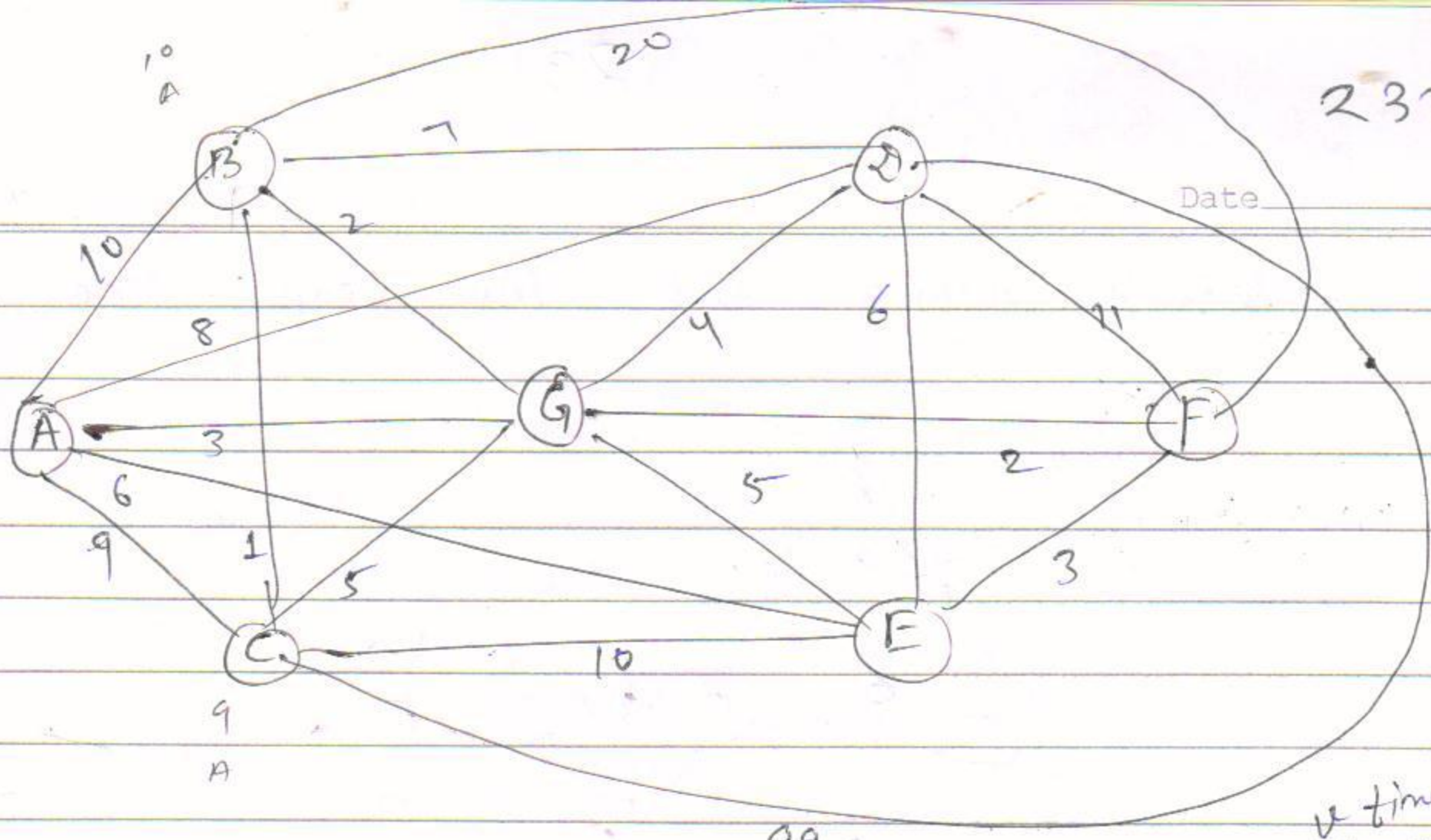




Edge sequance ⇒

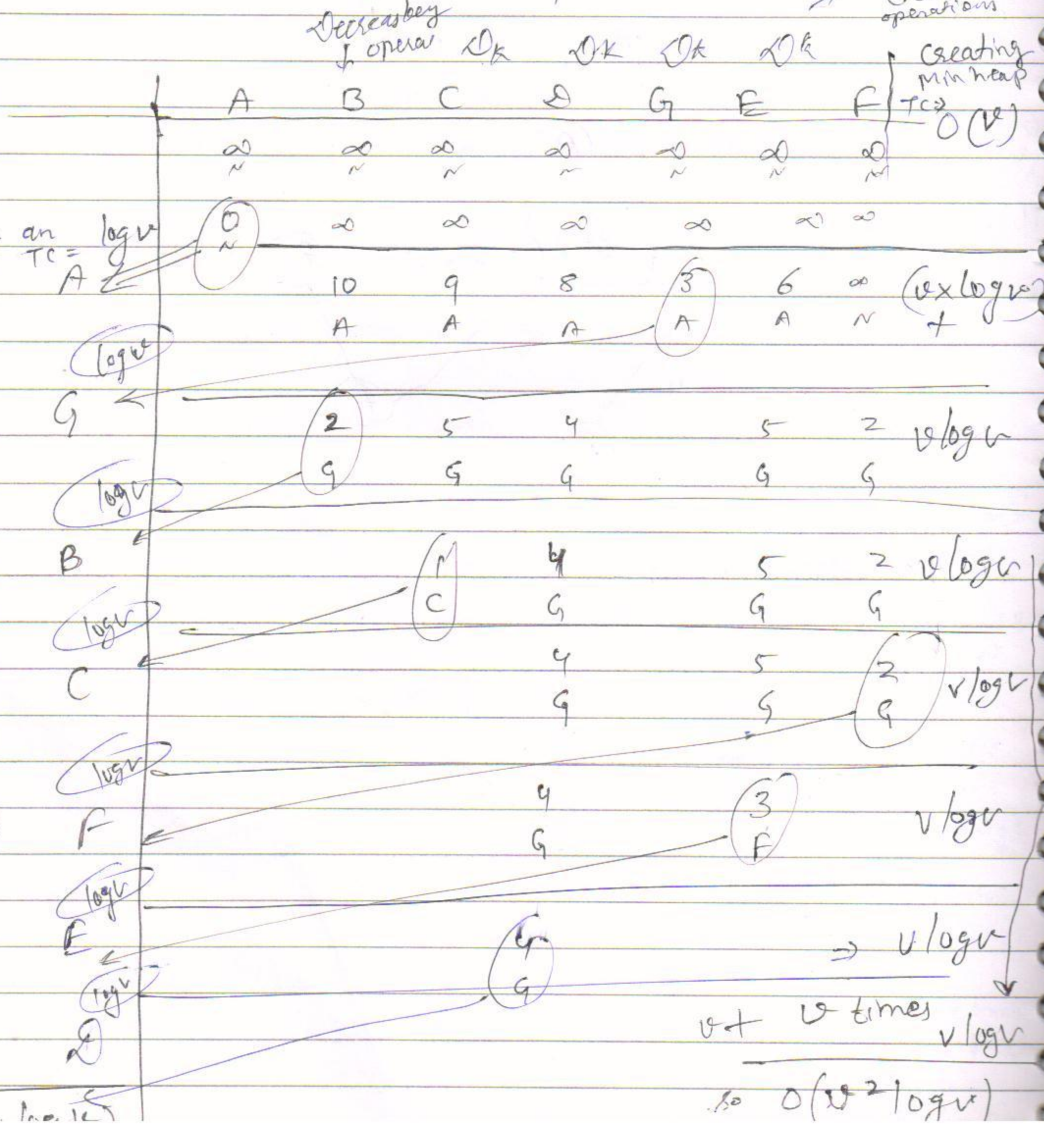
Gate

Q.



232

Date



T (Prims) $= v \log v + v^2 \log v + V$ — small

$= v \log v + v^2 \log v$

$= (v + v^2) \log v$

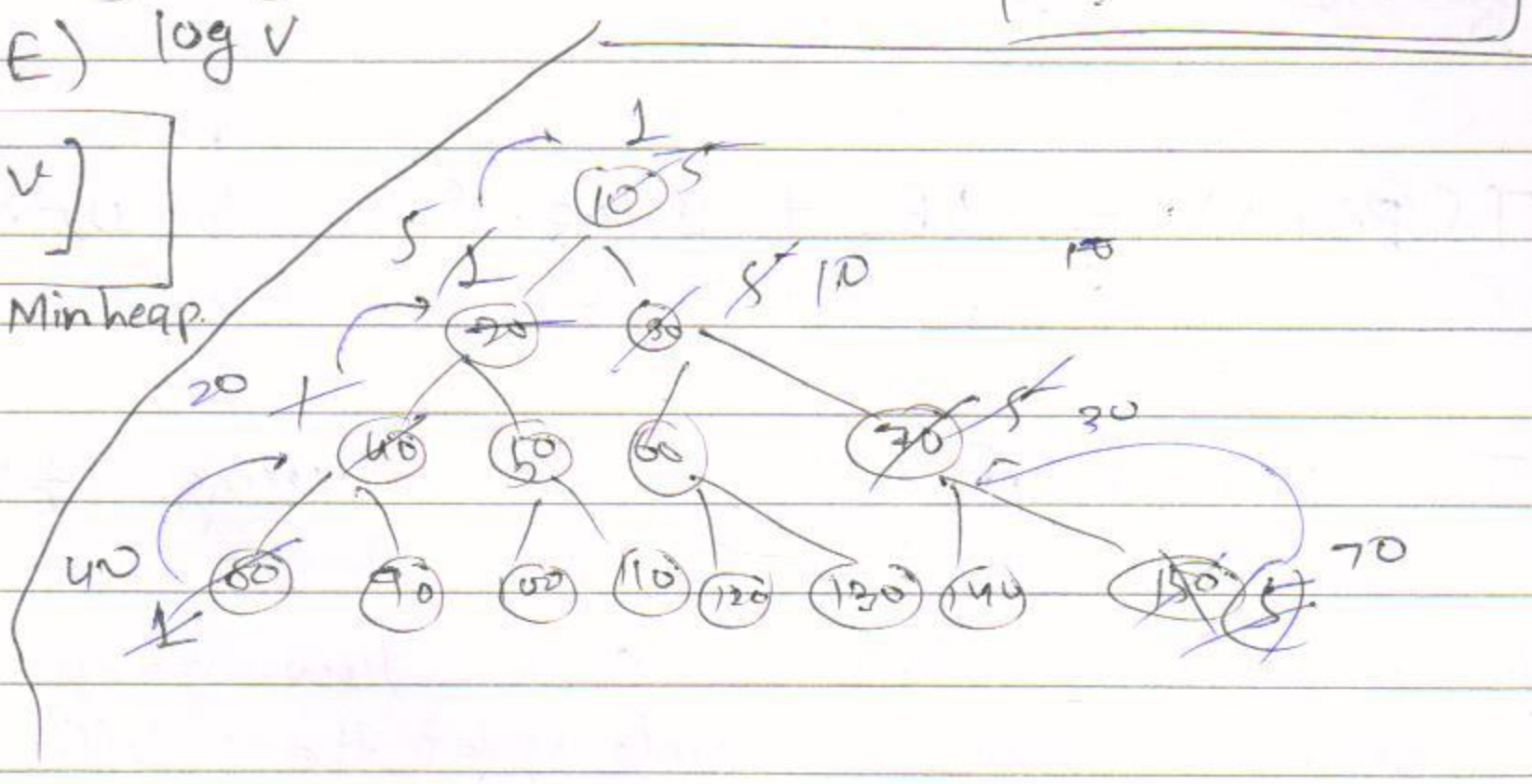
$= (v + E) \log v$

Date

$\therefore v^2 = E$

$T = O[(V+E) \log V]$

↳ using Binary Minheap.



go to place

↓

Decrese key (9, 15, 5) → replace 15th place by 5

or Decrese key (9, 8, 1)

} worst case ⇒ (log n)

or Decrese key (9, 12, 11) → best case no chang

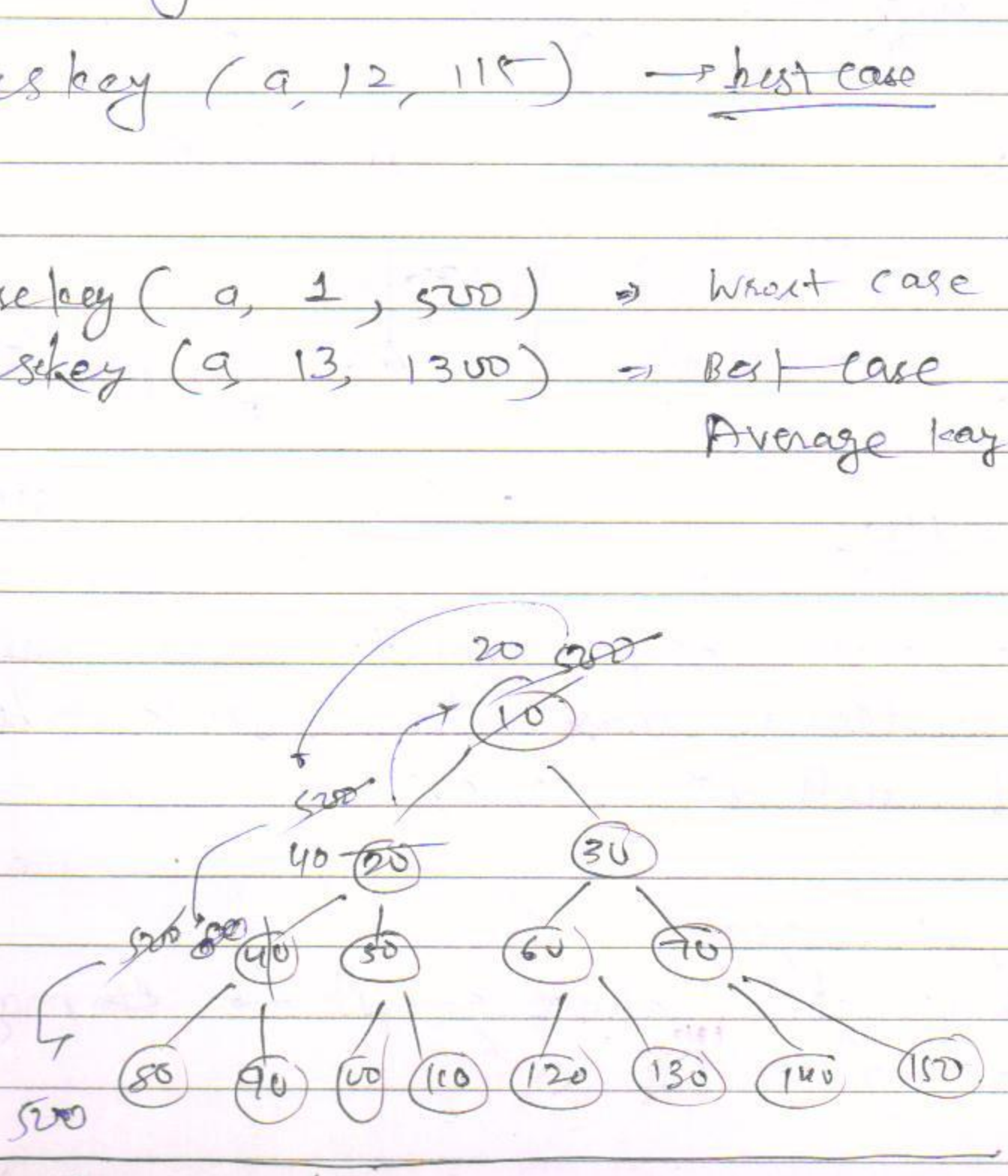
O(1)

Now

Increase key (9, 1, 500) ⇒ Worst case ⇒ log n

Increase key (9, 13, 1300) ⇒ Best case ⇒ O(1)

Average key = log n



→ Decrease key and increase key function in Minheap

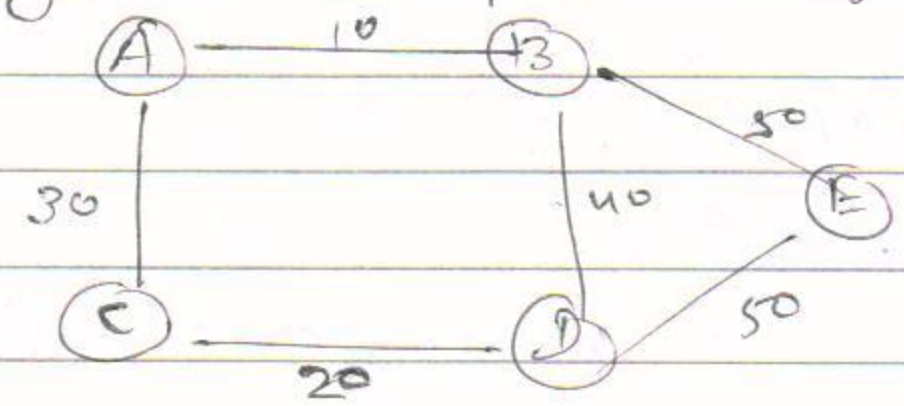
Thus finally:

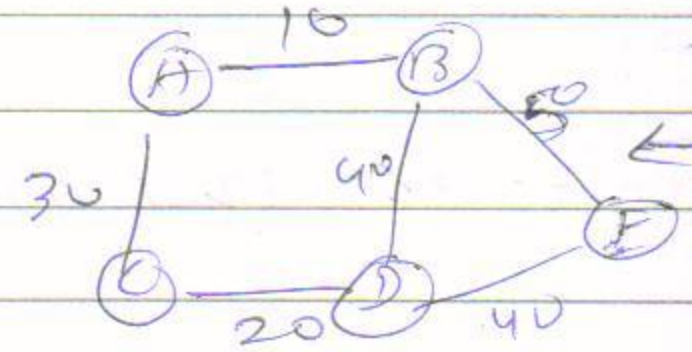
Time Complexity of (Prims) $= O[(v+E)\log v]$

$T(\text{Prims}) = O(E + v \log v)$ } using Fibanacci Min heap

$TC(\text{Prims}) = O(E+v)$ } using binomial Minheap.

Gate consider the following graph. How many minimum cost spanning tree possible for the following graph.

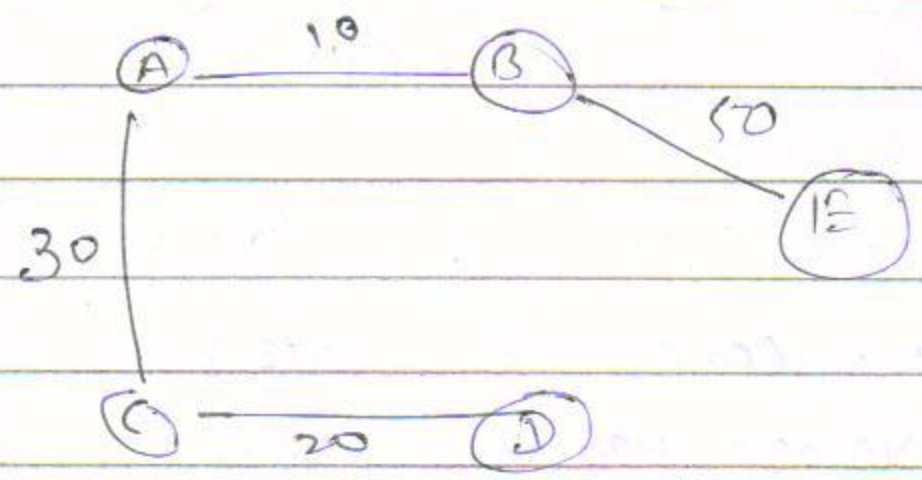




Ans.

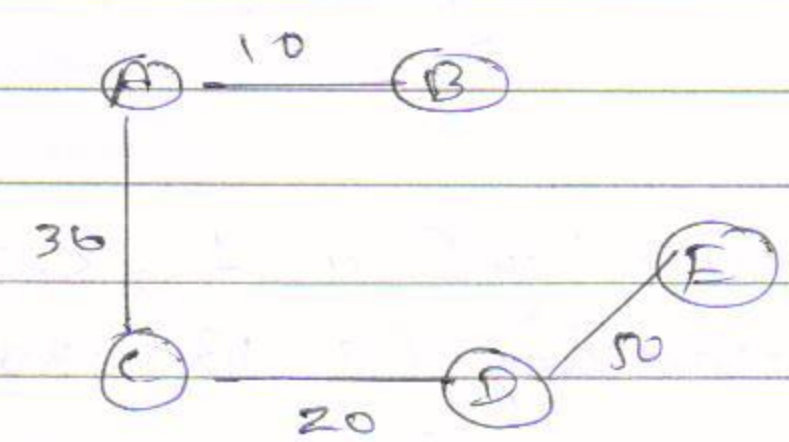
multiple answers.

① Ans



Cost = 110

Ans ②



cost = 110

Note:- For the given graph multiple minimum cost spanning trees may be possible but minimum cost will be same.

Q. Consider a given graph G,

→ If nothing is given about graph → may be no spanning

Imp ★★★

Date

(1) Edge Repeatition ✓

↓ ✗

Multple Minimum cost Spanning tree

→ ✓ given graph have edge Repeation. ↑ ✓

Multiple Minim C S T.

(2) G ⟹ 0 - MST (Disconnected graph)

(2) G ⟹ only 1 - MST (It is a connected graph and all the edge weights are distinct.) ✓✓

(3) G ⟹ more than 1-MST (Connected graph and repeated edge are there in graph.

★★★ Gate

(1) Let G be a Undirected Connected Graph with distinct edge weight. ✓

Let emax be the edge with maximum-edge weight & emin be the edge with minimum edge weight.

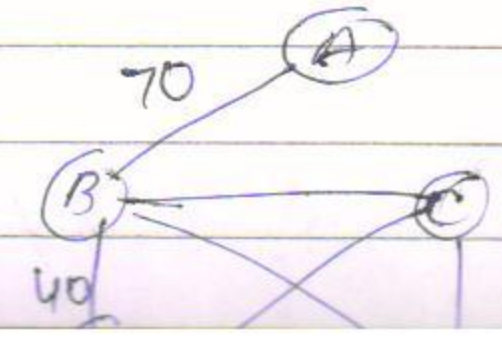
⇒ T & F ?

(a) ✓ G - contain Unique - Minimum Cost Spanning Tree (U-MST)

(b) ✓ G (MST) should contain emin

(c) ✗ G (MST) doesn't contain emax ✗

(d) ✓ If emax is in MST then it's removal from G, must disconnect G.



Q. Date

Let G be a undirected Connected graph (U-CG). Let w be the minimum edge weight among all edge weight. And E be a specific edge with weight w.

T.F.?

distinct not given

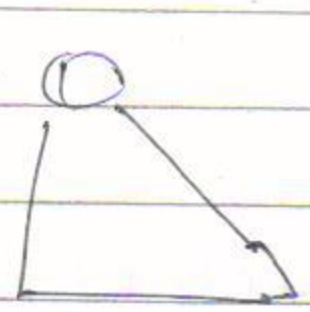
① e should be there in every MST. ✗

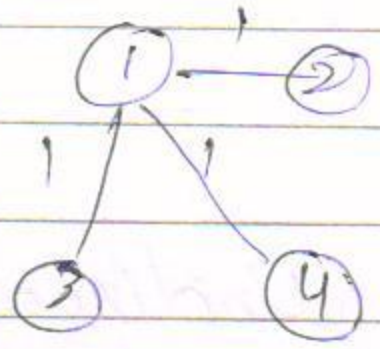
② e may be there in some MST. ✓

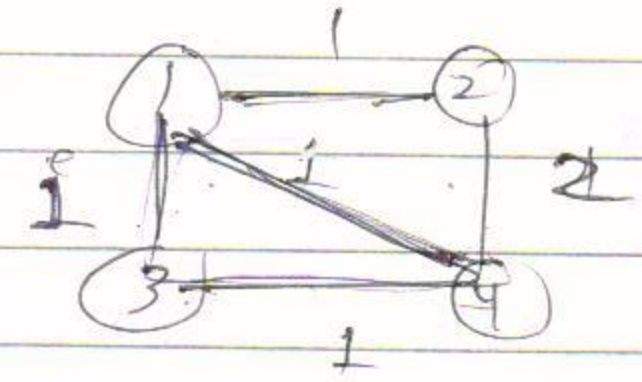
③ There should be a some edge with weight w in every MST. ✓

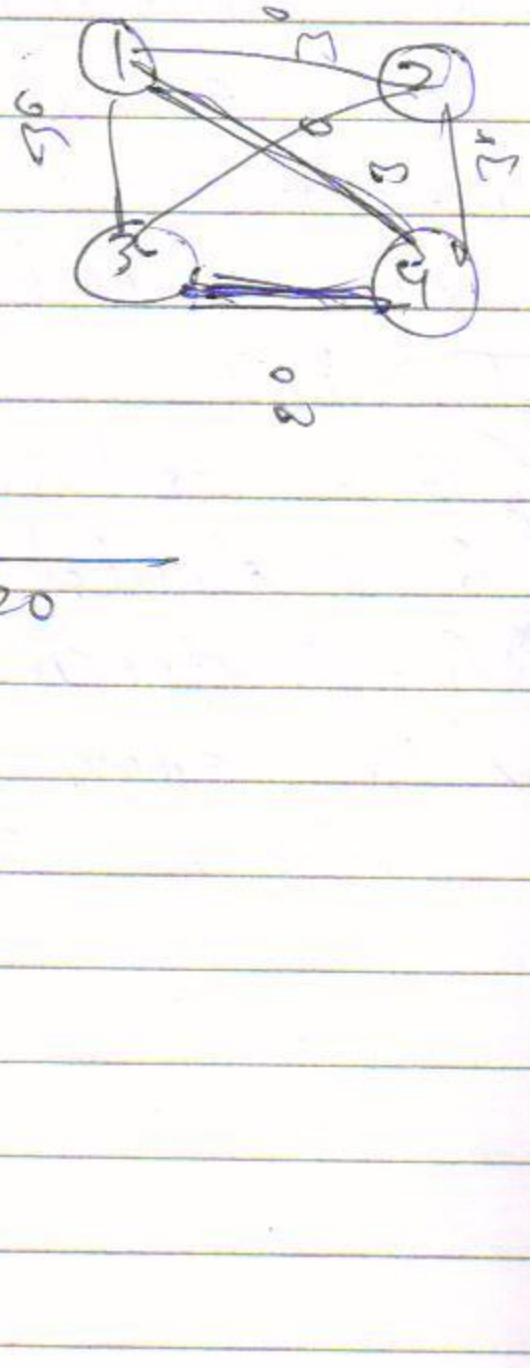
④ If e is not in MST then in that cycle all edges should have weight w. ✓

other









20

Date

Q. Let G be a UCG, and T and T' are 2 spanning trees of graph G.
e be a specific edge present in T but not in T'.
&&
e' be an edge present in T' but not in T.
and assume all other remaining edge should be same
then what will be T and T' after performing following operations —

(i) $(T - \{e\}) \cup e'$

(ii) $T' - \{e'\} \cup e$

T.F →

(a) ✓ both T and T' are spanning trees. — ✗.

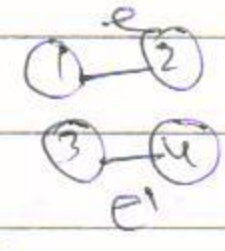
(b) T is spanning tree but T' is not.

(c) T' is spanning tree but T is not.

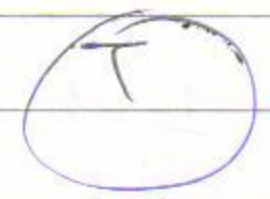
(d) both T && T' are not spanning tree.

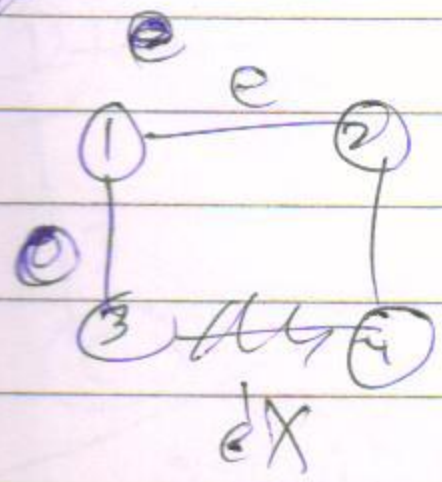
ans.

connected graph G

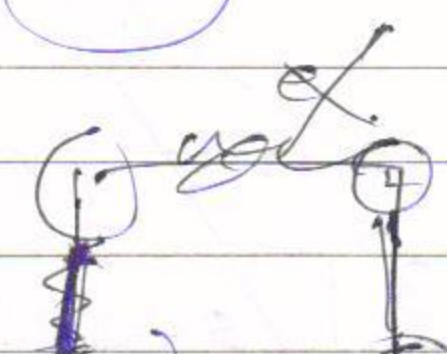


Spanning tree





✗



spanning tree

$T - \{e\} \cup e'$

(ii) $T' - \{e'\} \cup e$

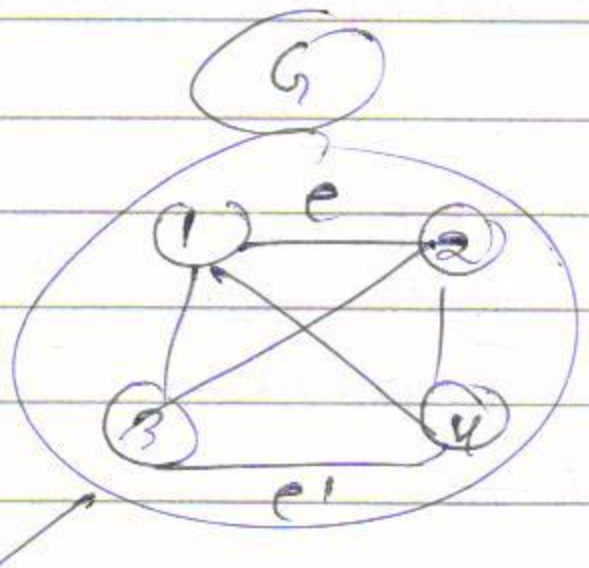
1 2 become

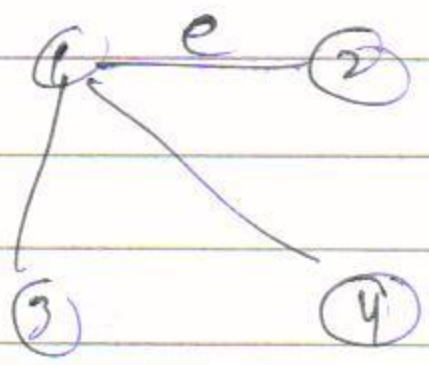
spanning tree

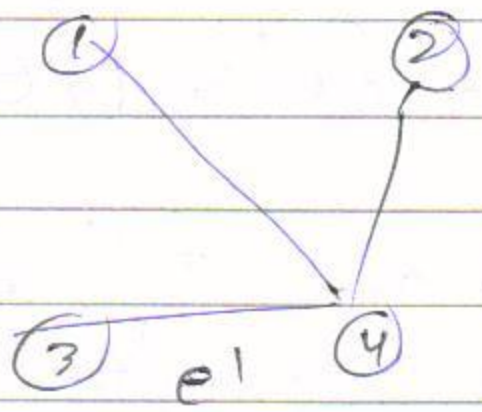
Date

If that line in previous questions - ~~Also~~ is not these ("and all other remaining edges are same").

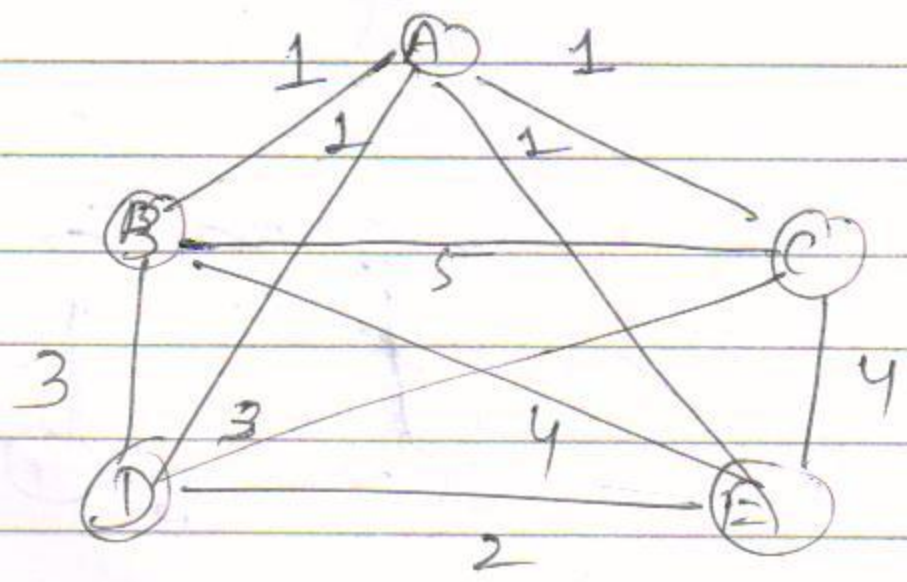
then







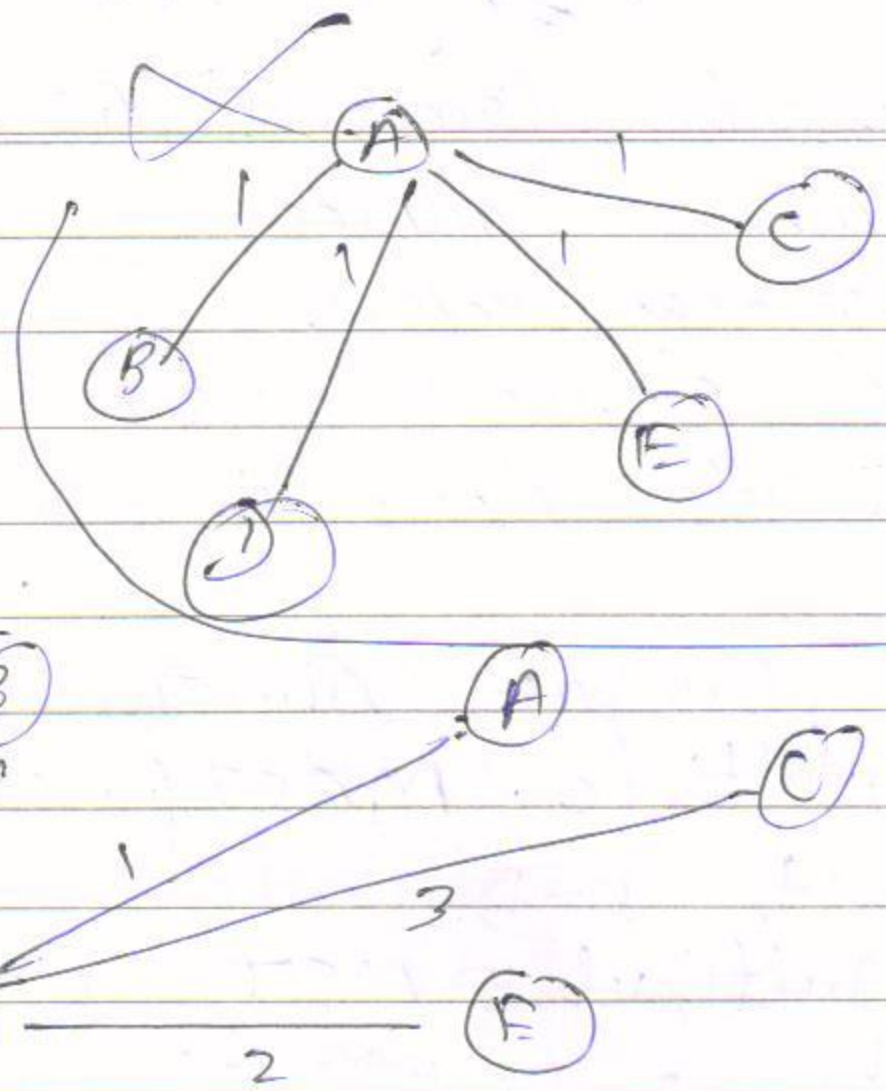
Q. Consider the following graph. -



What will be the cost of minimum cost spanning tree for the above graph so that vertex A is leaf node in that spanning tree.

Date

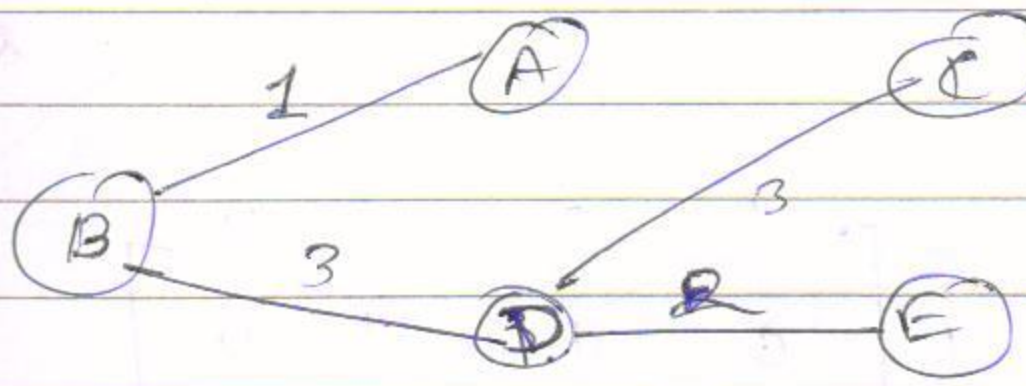
Ans.



= (4) ✗ but here (A) is not leaf node here

✓ Ans. - 9

or, go for prims algo and choose anyone beside A

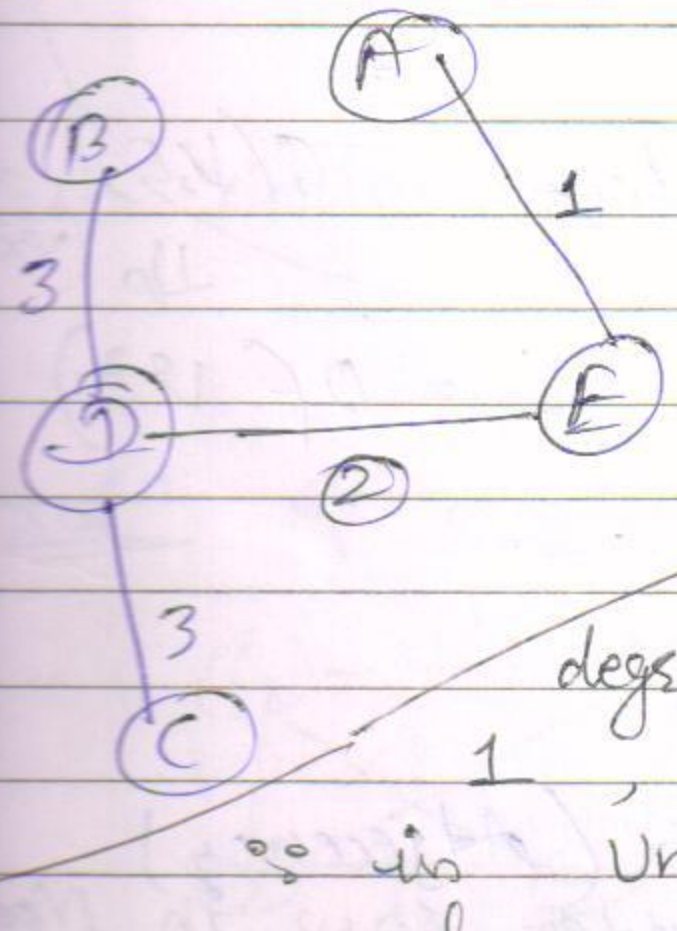


- leave A again

= 3 + 1 + 3 + 2

= (9)

let root is (E) A



⇒ 3 + 3 + 2 + 1

⇒ (9)

can start from anywhere but degree of A should not be more than 1, bcoz it is undirected graph ∵ in undirected graph leaf node have degree 1.

Q. Let G be a Undirected Connected Graph (UCG), with n vertices, whose adjacency matrix is given by n×n square matrix in which
(i) all diagonals element are 0
(ii) all non-diagonal element are 1

T/F ?

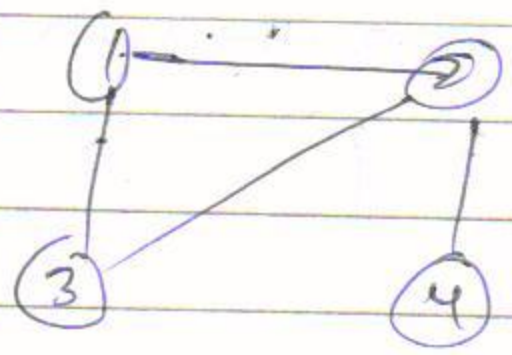
(A) ✗ Graph G is Uunique Minimum Cost Spanning Tree (UMST) [No distinct so false]

(B) ✓ G - contain ~~M~~ multiple MCST (MST) each of having cost (n-1)

(C) G - contain Multipul MST each of different cost → no way → [No two MST spanning tree for one graph have same cost]

(D) ✗ ~~G - contain~~ MST. G don't have MST. [Not disjoint so false]

Ans.



⇒

	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	0
4	0	1	0	0

Adjacency Matrix G(V,E)

I/p (V×V)

[Space S.C. = O(V²)]

*Basics ⇒

G —— (10, (0000)

↓

Need of matrix ⇒ 10×10

⇒ degree of any vertex if matrix (Adjecency) — count all ones of that vertex row in Matrix

No of edges ⇒ all ones (Sum of ones) in matrix / 2

⇒ Note :- In graph theory if any edge is given b/w a vertex and edge weight is not given then by defaut edge weight = 1

G (1000, 0)

(241)

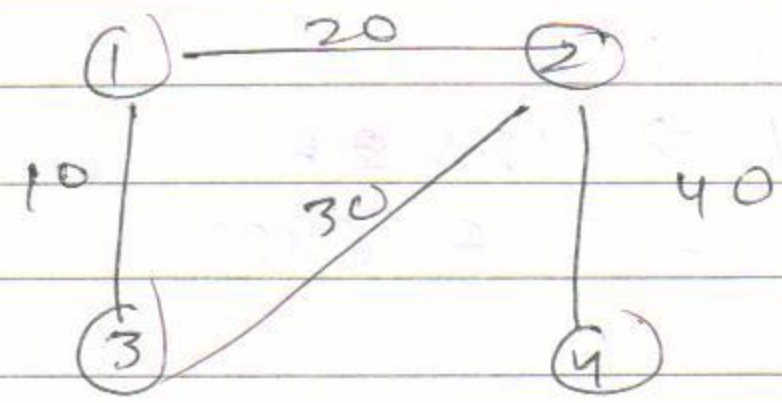
↓↓

space required → (1000^2)

Ans. // (Adjacency Matrix method is good for Null graph (in which vertex may be any but all vertex having 0 degree means Not having ~~sin~~ single edge.)

But this method is very bad for → Complete graph.

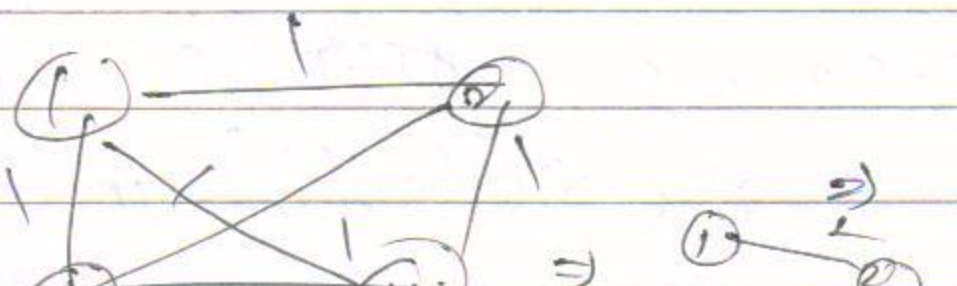
2nd Method:



Cost Adjacency Matrix =

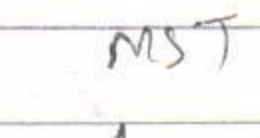
	1	2	3	4
1	0	20	10	0
2	20	0	30	40
3	10	30	0	0
4	0	40	0	0

Sol^n of ~~Ans~~ Question :-



0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

MST



by defaut edge weigh is → (1)

Let n x n

4 x 4 = (n-1) = 4-1 = 3

cost = (n-1)

3 = 1+1+1 = (n-1)*1

⇒ Cost is always (n-1)*(1)

Date

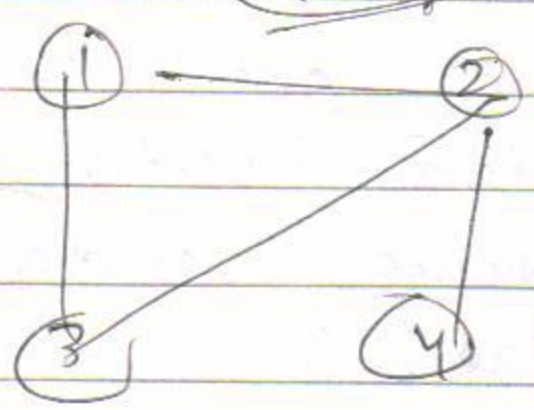
⇒ Adjancy - list :-

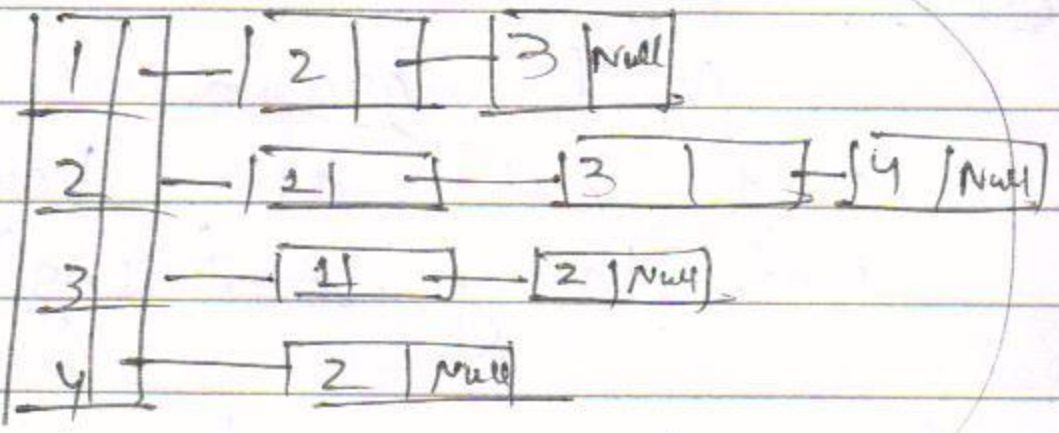
keep adjency element, keep in link-list is called adjacency list.

if keep in matrix is called ~~Mat~~ Adjacency matrix.

4 edges → 8 edges

eg:-





Complexity = array of total vertex (v)

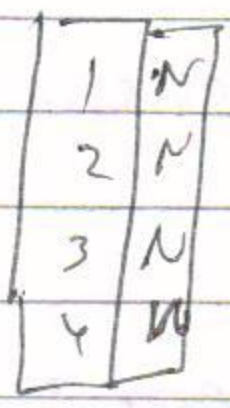
+ double of edges (2E)

* Good for less edges

* Bad for More edges ⇒ $v + 2E$

$= \theta(v)$

Adjucency list if - Null graph ⇒ No edges.



$TC. = O(v)$

complete graph ⇒

for every node/vertex ⇒ (v-1) eges.

$\Rightarrow [2(v-1)]$ in

so all v vertexes ⇒ $v \times (v-1)$

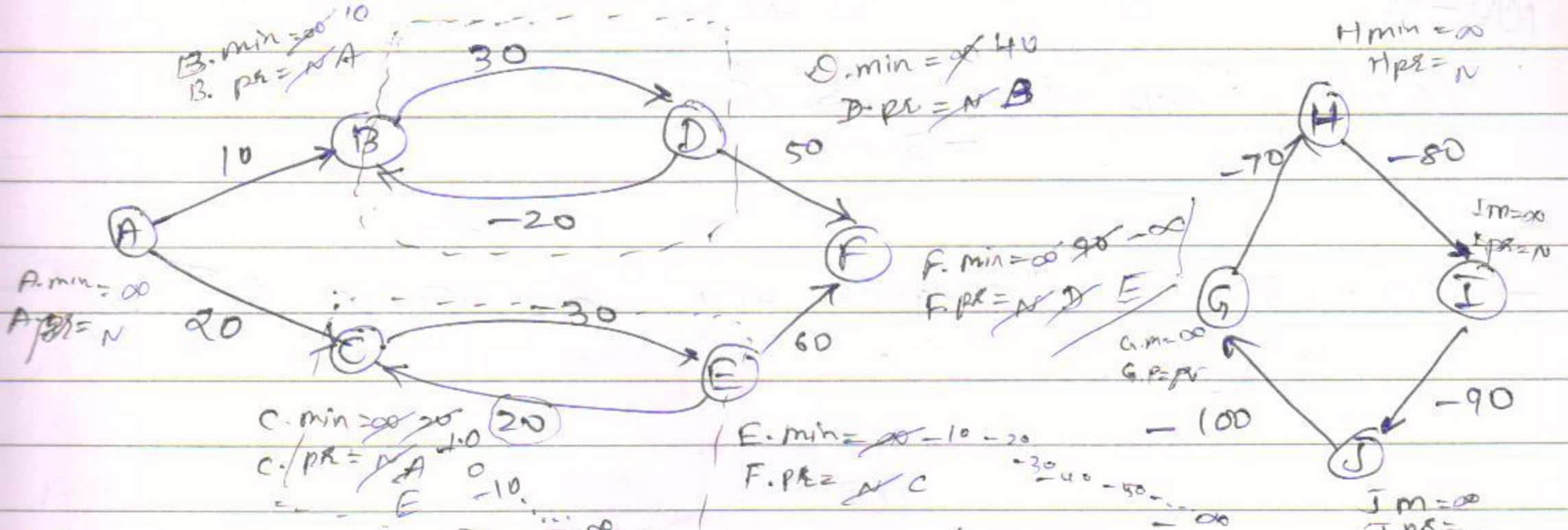
$= O(v^2)$

If anyone says → For Any Problem dynamic Programming is best, means, Dynamic programming giving right Ans. or taking less time with comparision of greedy Tech.

⇒ Single Source shortest Path (SSSP):— Date

243

from one source to all other vertex, minimum cost.



Goal ⇒ A to every vertex find minimum cost.

eg Like

Fmin ⇒ Minimum cost to go from A to F

D is call B and say i am coming from A and my cost is 40 and i can reach to you (B) in $ (40 − 20) = 20, but

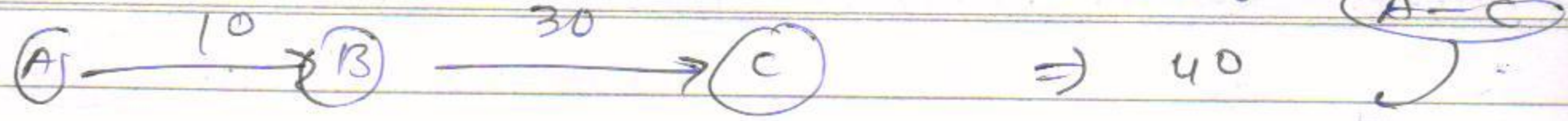
There are 2 cycles —

1st cycle ⇒ (30)+(−20) ⇒ 10, always increas 10 ...

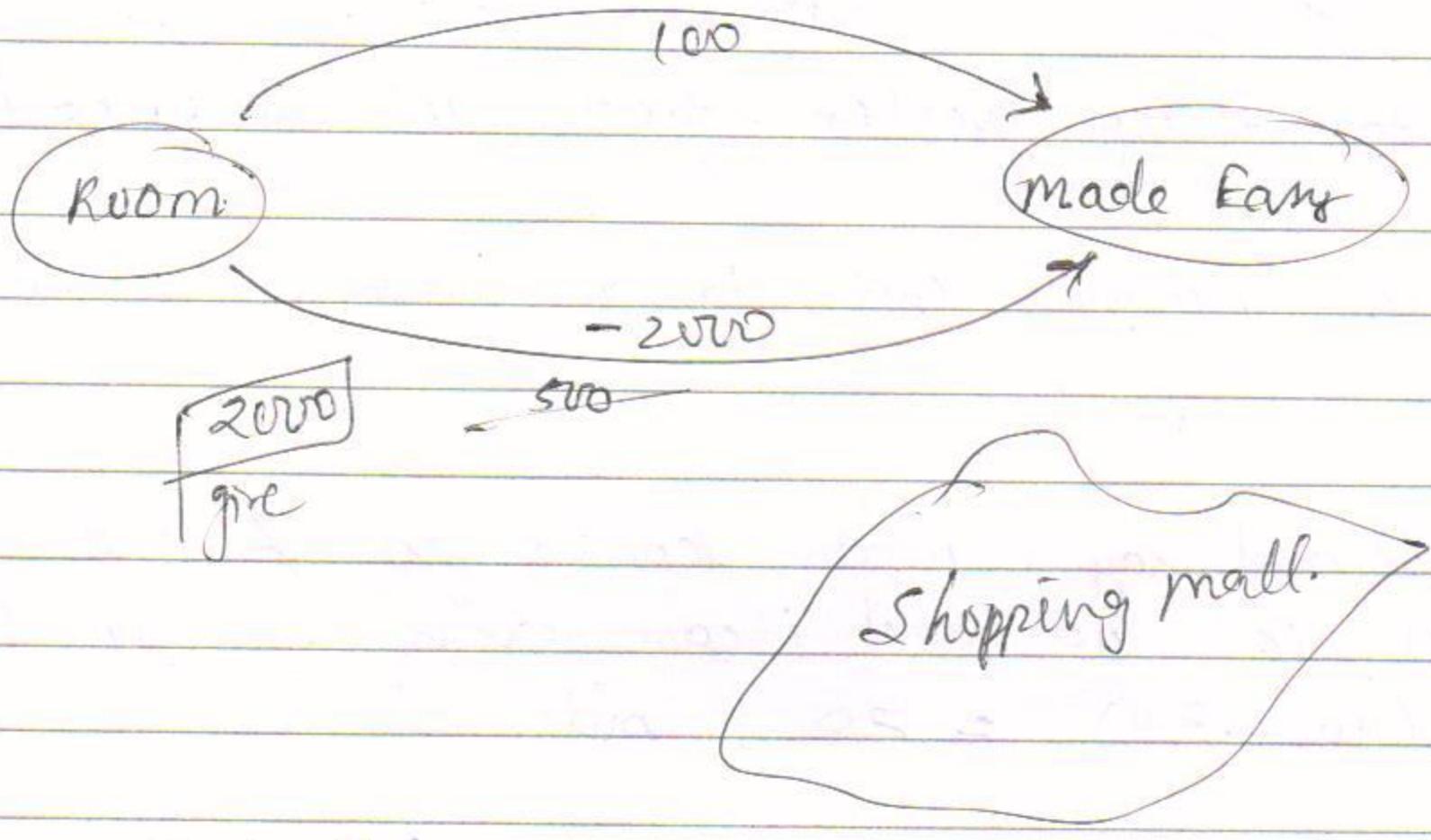
2nd cycle ⇒ (−30) + (20) ⇒ −10 → always decreas −10 −10...

and finaly becom −∞.

⇒ The C, E & F, vertices are get −∞ bcoz they are either part of the (−)(negative edge weight or depend on negative edge weight part.



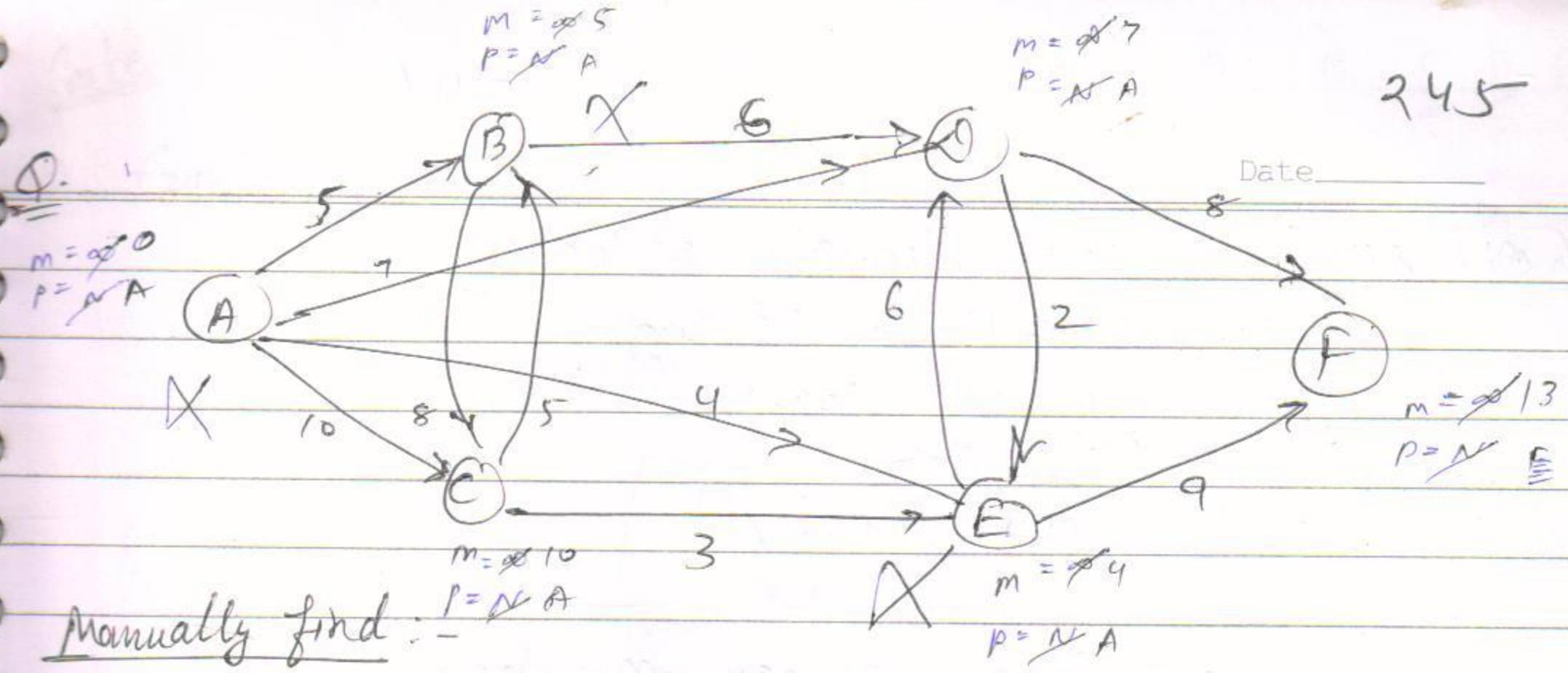
→ conductor will give you 30 Rupees back.
means (−)



⇒ Min Cost (A, B) = ∞ [No Path]

⇒ Min Cost (A, B) = −∞ { B is past of −ve edge weight cycle (or) Dependency upon the edge weight cycle }

Date



Manually find :-

A — A =	0	✓
A — B =	5	✓
A — C =	10	✓
A — D =	~~6~~ 7	✓
A — E =	4	✓
A — F =	13	✓

Whenever same cost will come don't change

Using Dijkstra's Algo :-

	A	B	C	D	E	F = 0 (V)
	0 N	∞ N	∞ N	∞ N	∞ N	∞ N
log V A		5 A	10 A	7 A	4 A	∞ N
log V E		5 A	10 A	7 A		13 E
log V B			10 A	7 A		13 E
log V D			10 A			13 E
log V C						13 E
log V F						
V time						V log V

best people via A & E

$A^x E^x B^x$

$O(V \log V)$

$(V \times V \log V)$

Date

T (Dijkstra) $= v + v \log v + v^2 \log v$

$= v \log v + v^2 \log v$

$= (v + v^2) \log v$

$T = O[(V+E) \log V]$

↳ using Binary min heap

$T = O(E + V \log V)$

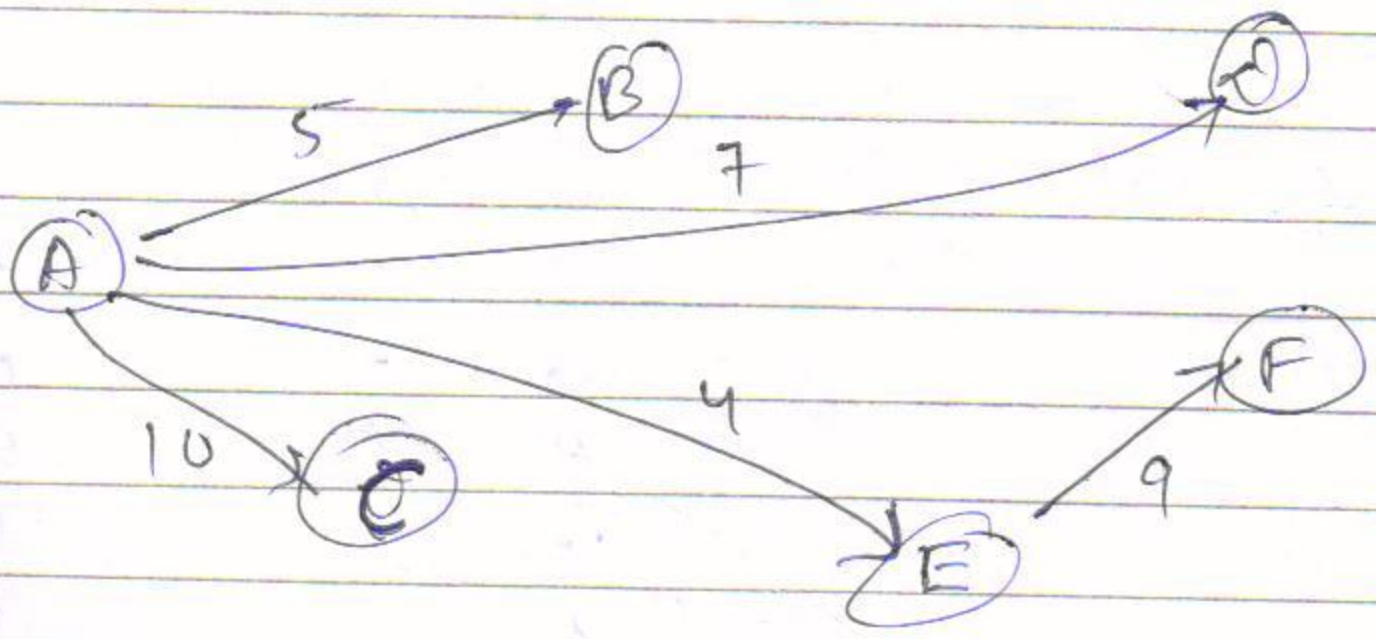
↳ using Fabonacci Min heap.

$T_{Di} C = O(V+E)$

↳ using Binomial Minheap.

Finally MST:-

Sol^n

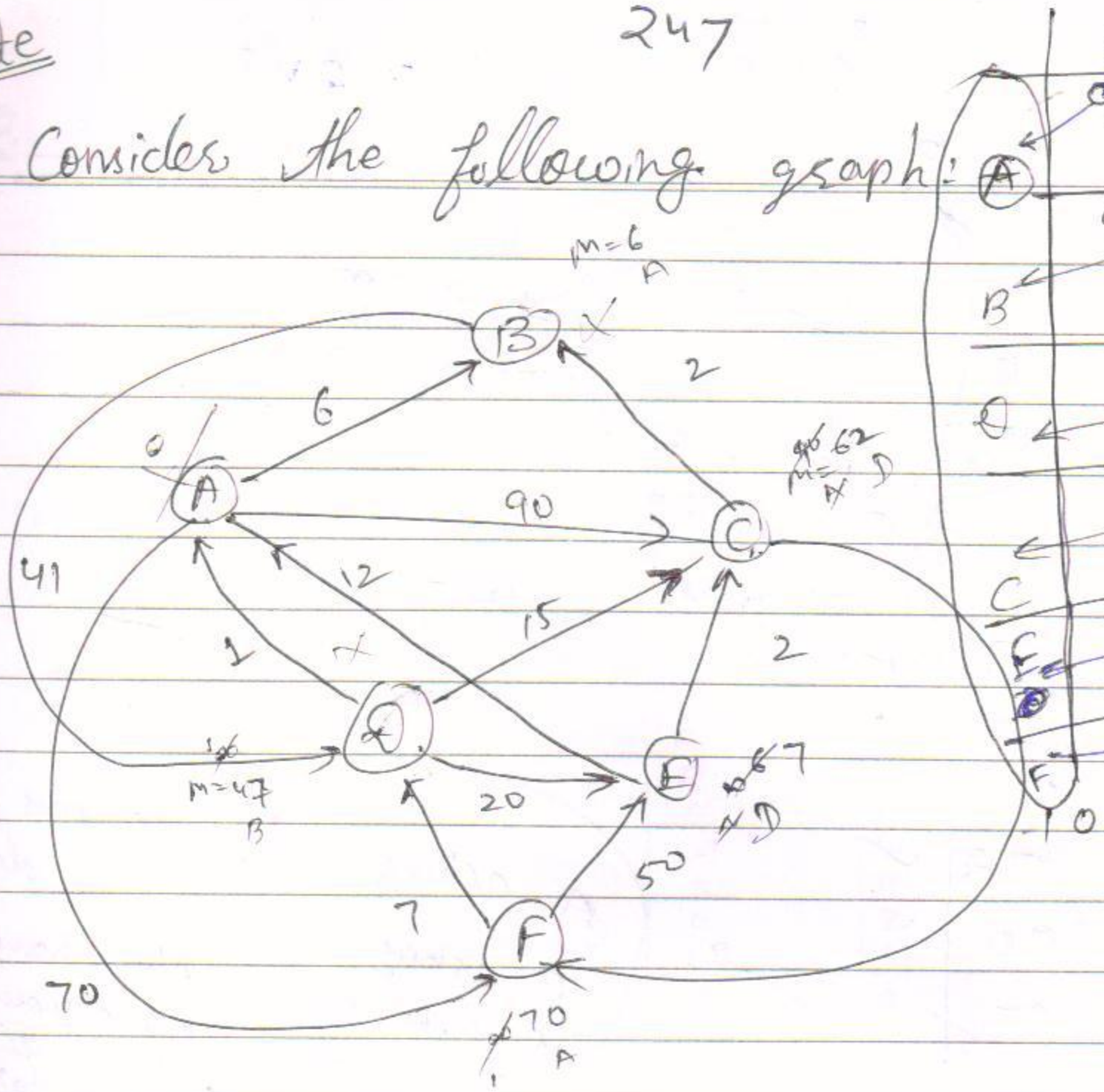


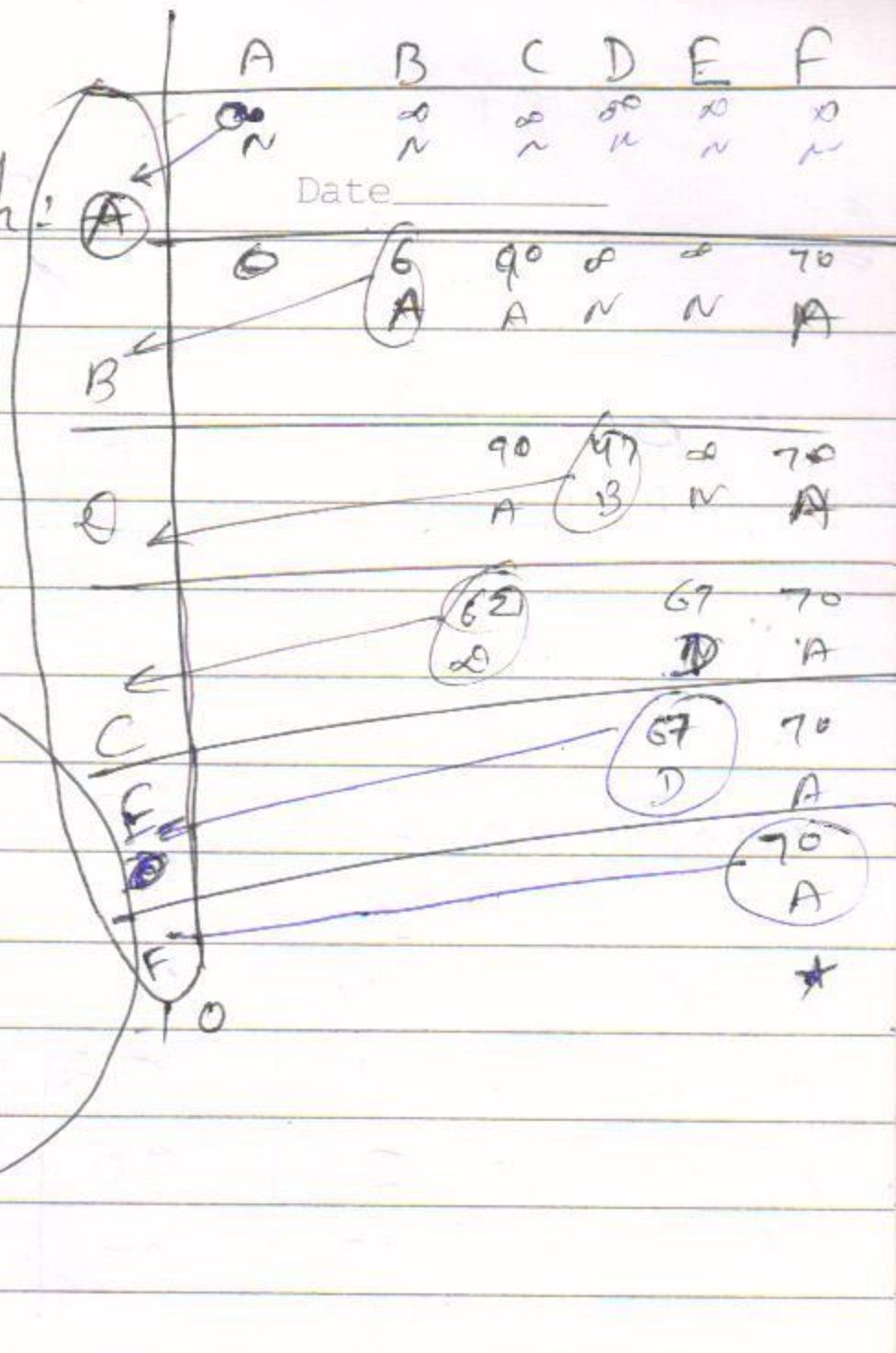
Q- in which order single source sortest path.

A - E - B - D - C - F.

Gate

Q. Consider the following graph:





Q.1) Output the sequance of vertices ~~identi~~ identify by the dijkstra when the algo is started from the algo A.

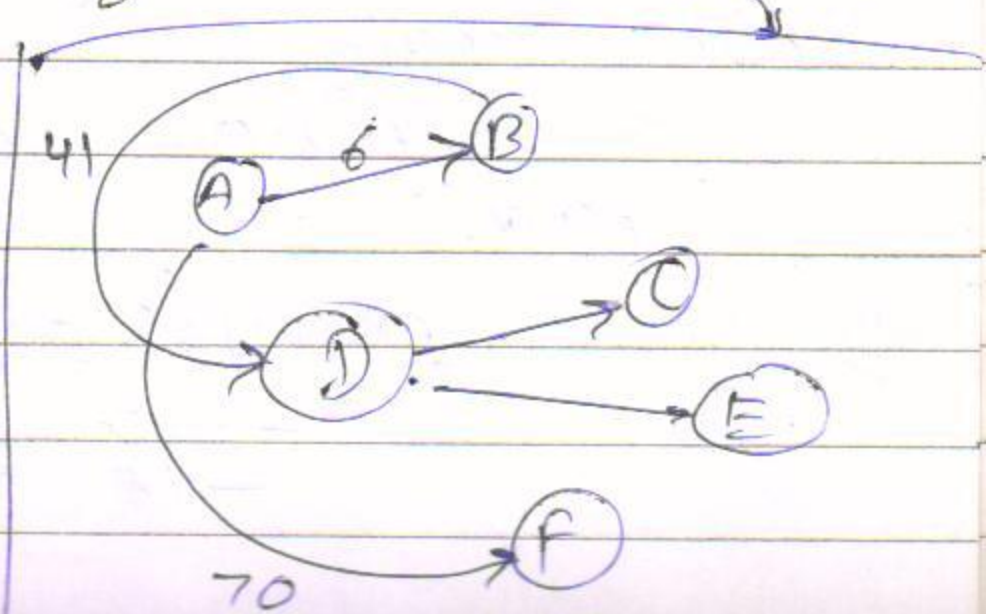
Ans | A - B - D - C - E - F |

Q2 Output the ~~sequance of~~ vertices shorted path from A to E.

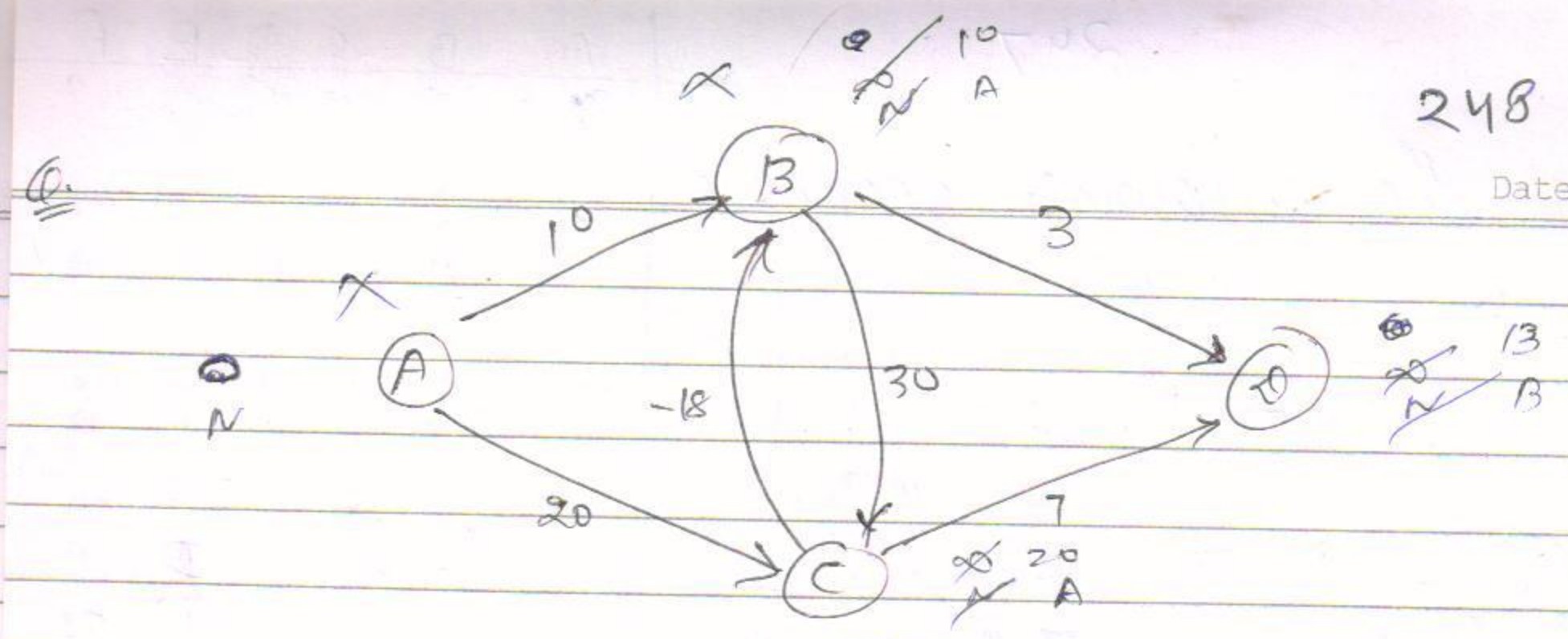
Ans A — B — D — E

Q3 What is the Minimum Cost from A to F

Ans 70 Ans



Date



Manually do ⇒ consider (-) nodes

A – A	⇒	0	0
A – B	⇒	10	2 ✓
A – C	⇒	20	20
A – D	⇒	13	5 ✓

A asking why are you deleting
B asking why are you deleting
Dijkstra said, No chance to reduce your cost
But Bellman ford give concept of (-) negative cost

Now :- Dijkstra :-

A-A ⇒ 0 ✓
A-B ⇒ 10 ✗
A-C ⇒ 20 ✗ ✓
A-D ⇒ 13 ✗

	A	B	C	D
A	∞ N	∞ N	∞ N	∞ N
B	10 A	20 A	∞ N	∞ N
D	20 A	13 B	∞ N	∞ N
C	20 A	13 B	∞ N	∞ N

to Dijkstra

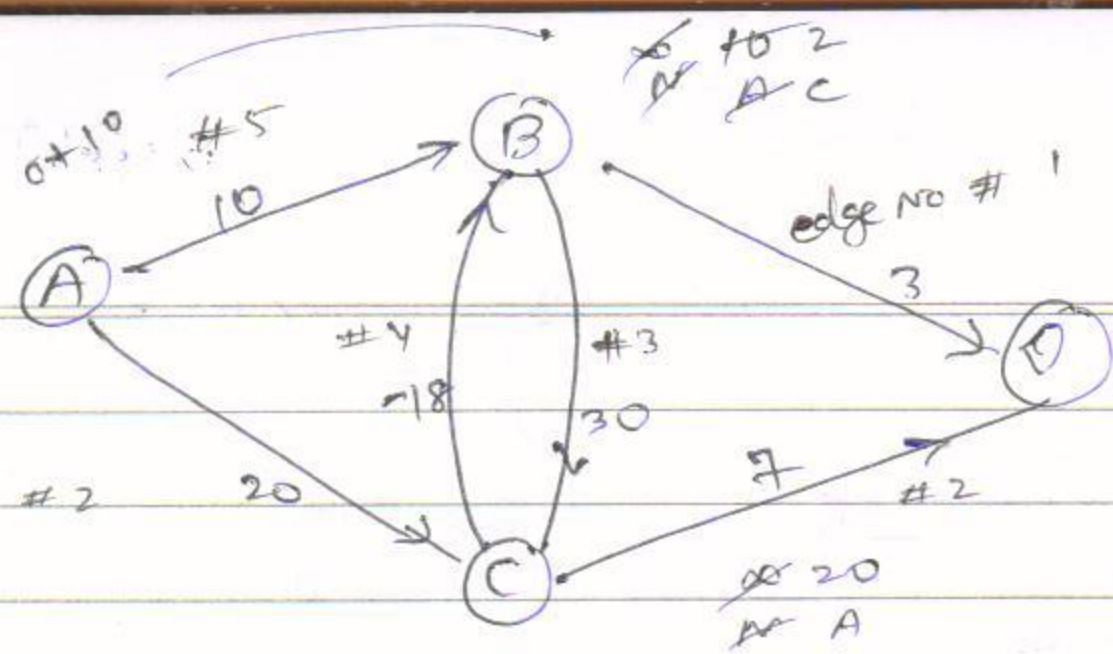
⇒ Bellman-ford :- Bellman ford says ^ that why are you deleting, you may need to change value again.

so Bellman don't delete and keep all vertex.

in above example –

here C ⟶ B (−18)

Date



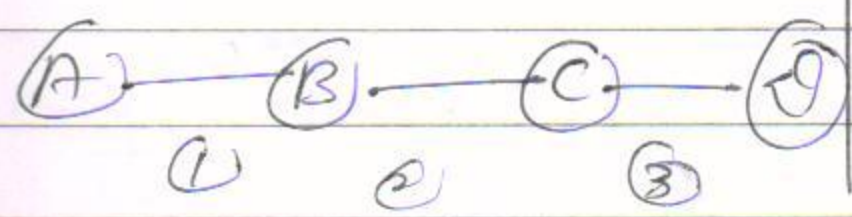
Relaxation on all edges one time is one pass.
we have to do v - passes.

	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	10	20	∞
	N	A	A	N
i=2	0	2	20	13
	N	C	A	A
i=3	0	2	20	5
	N	C	A	B
i=4	0	2	20	5
	N	C	A	B

→ No change checking purpose

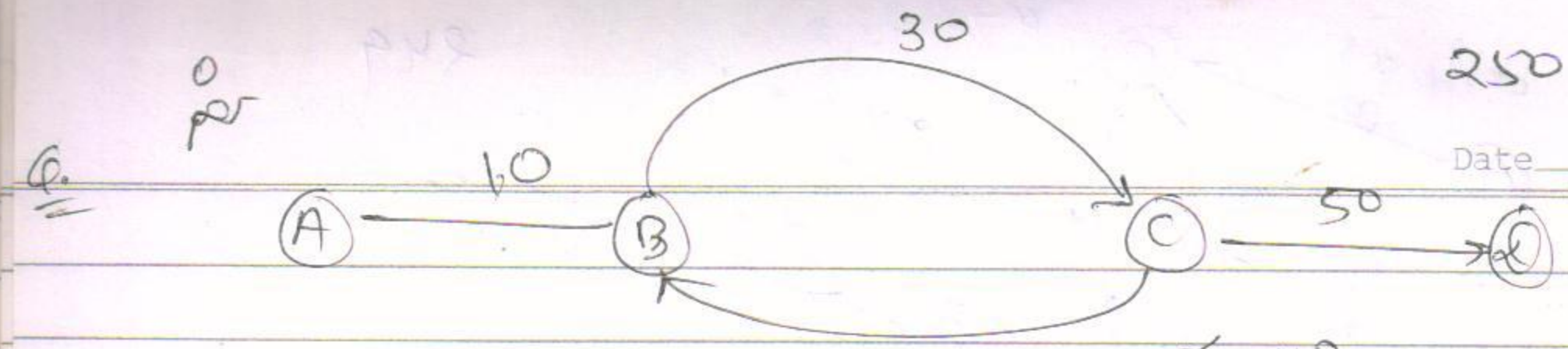
why only 3 pass are sufficient.

Ans What is the largest path possibe with (4) vertics.



A - A	→ 0
A - B	→ 2
A - C	→ 20
A - D	→ 5

∴ 3 is sufficient.



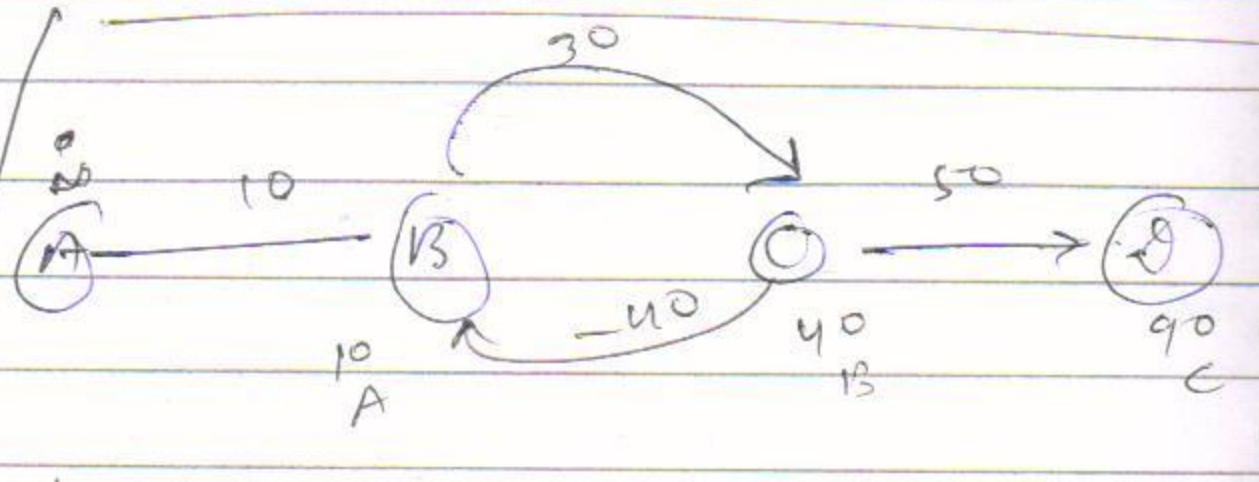
Manually :-

A — A ⇒ 0 ✓

A — B ⇒ ~~10~~ -∞

A — C ⇒ -∞

A — D ⇒ -∞



Dijkstra :-

A - A = 0 ✓

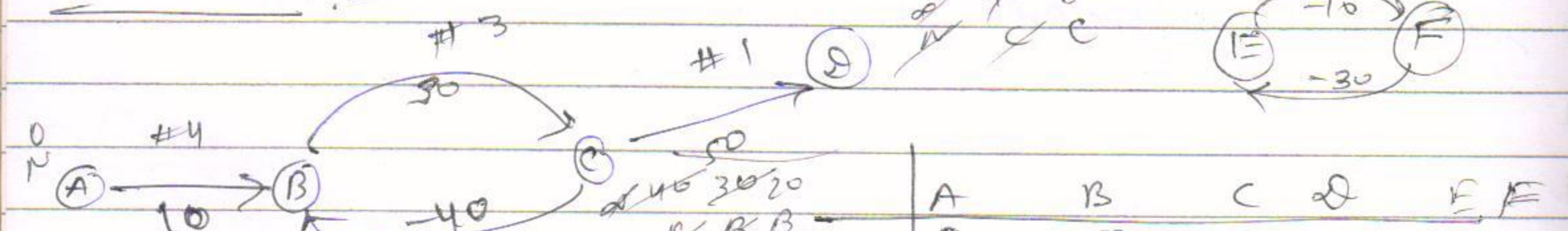
A - B = 10 ✗

A - C = 40 ✗

A - D = 90 ✗

	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
A		10	∞	∞
		A	N	N
B			40	∞
			B	N
C				90
				C
D				

Bellman Ford :-



	A	B	C	D	E	F
	0	∞	∞	∞		
	N	N	N	N		
i=1	0	10	∞	∞		
	N	A	N	N		
i=2	0	10	40	∞		
	N	A	B	N		
i=3	0	0	30	90		
	N	C	B	C		
i=4	0	-10	20	80		
	N	C	B	C		

Verification

always change → so negative cycle part or

Date

In

Bellman ford

Time complexity of Bellman ford algo is

$$\underbrace{E + E + E + - - - - + E}_{V \text{ times}}$$

$TC \Rightarrow O(VE)$

Q. Using Bellman ford can you find -negative edge weight cycle.

=> dn → dO i.e. ⇒ v or n time, # vertex = v

Q. Which one is true —

1. ~~X~~ Bellman ford find out all negative edge weight cycle. ✓
2. Bellman ford find out all negative edge weight cycle if graph is reachable from given (A) node.

=> Bellman ford algo will give correct answer if the graph contains ~~or negative ed~~ positive edge weight cycle or negetive eg

→ Dijkstra algo fails sometimes if graph contains negative edge weight.

→ Di algo fails always if graph contains -ve edge weight cycle.

→ Bellman ford Algo find all -ive edge weight cycle if it is reachable.

9-Nov-12

25 total Marks ⇒ 20 marks comes from the notes.

252

Date

⇒ Dynamic programming :-

Greedy Technique :- ① Sometimes gives wrong answers. Ex. In Greey. In the case of Dijekstra, (-ve) edge weight cycle

Dynamic Programming :- ① Always give correct Answers.

(DP.) ⇒

G T :- ② One decision at a time	③ Most probabily ~~More~~ Less time
DP :- ③ Seqence of decision	③ Most probabily More time.

⇒ Applications of Dynamic Programming :-

① Fibinocci Series :-

② ~~Longest~~ [Longest Common subsequences (L.C.S).

③ Matrix chain multiplication

4) Travelling seles person problem

5) Sum of subsete problem

6) 0/1 knapsack

7) All pairs shortest path.

(1) Fibinocci Series :-

n	0	1	2	3	4	5	6	7	8	9	10
fibin	0	1	1	2	~~3~~	5	8	13	21	34	~~55~~

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-1) + fib(n-2) & \text{if } n \geq 1 \end{cases} \quad \left\{ \text{if } n \right.$$

- Tree means, that program is recursive program.
- If binary tree is coming means, that block of programming is

$$T(n) = T(n-1) + T(n-2) + C$$

Recursive program of fibnocci

Date 253

```
fib ( n)
{
  if ( n==0 || n==1)
    return (n)
  else
  {
    a = Fib( n-1)
    b = Fib( n-2)
    c = (a+b)
    return (c);
  }
}
```

$T(n-1)$

$T(n-2)$

C

2 function are there in else part

Make (draw) recursive tree for

Let $n=6$

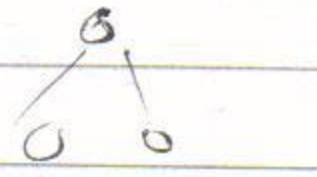
$O(n^2)$

Date

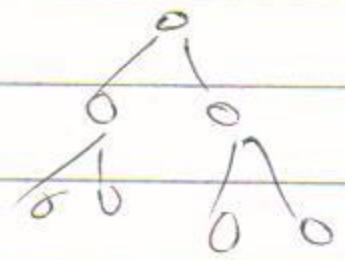
$n * (n-1) * (n-2) * (n-3) \cdots * 2 =$

$n * n * n * n \cdots * n = O(n^n)$ Worst

$2 * 2 * 2 * 2 * 2 \cdots * 2 = \Omega(2^n)$ best.

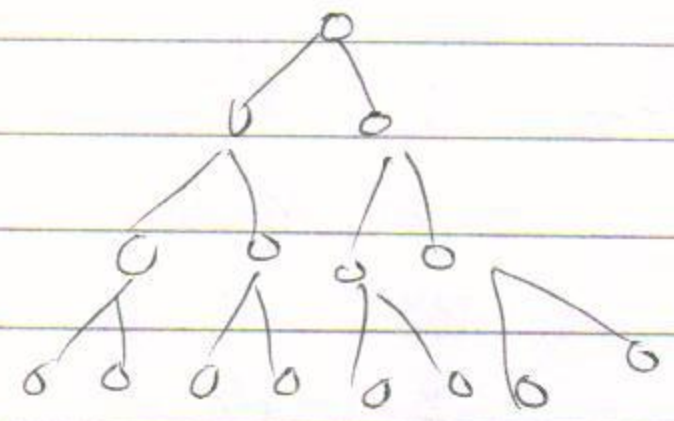


$\rightarrow 2^2 - 1 = 3$ nodes



3 level.

$\Rightarrow 2^3 - 1 = 7$



$2^4 - 1$

n levels $2^n - 1$

Fib (n) = n - level Complete Binary tree

$= 2^n - 1$ nodes

$= 2^n - 1$ function calls

$= 2^n *$ very function take constant time

$2^n * O(1)$

$= O(2^n)$

OR $\Rightarrow$

$= T(\text{Fib}(n)) = 2^0 c + 2^1 c + 2^2 c + 2^3 c + \cdots + 2^{n-1} c.$

$= c[2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{n-1}]$

$= c\left\{a\frac{(r^{n-1})}{r-1}\right\} = c\left[1\frac{(2^{n-1})}{2-1}\right)$

Date

$\Rightarrow c \cdot 2^n$

$T(fbin(n)) \Rightarrow O(2^n)$

if →

else

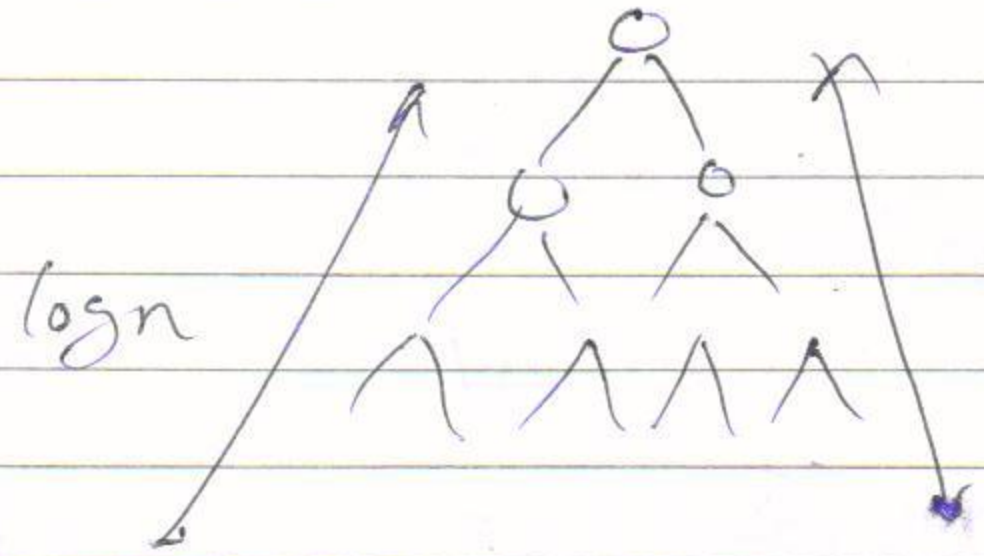
{

$a = fun(n/2)$;

$b = fun(n/2)$;

$c = a+b$;

}



or

$T(n) = T(n/2) + T(n/2) + c$

$= 2T(n/2) + c$

master theorem

$= n\_level$

$= 2^{\log n} - L$

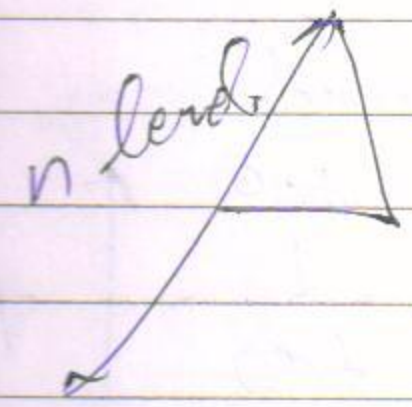
$= 2^{\log n}$

$= n * c$

$= n(a(1))$

$= n$

$T(n) = T(n-1) + T(n/2) + c$



$= n\_level.$

$= (2^n - 1)c$

$= 2^n \cdot c$

$= 2^n\, a(1)$

$= O(2^n)$

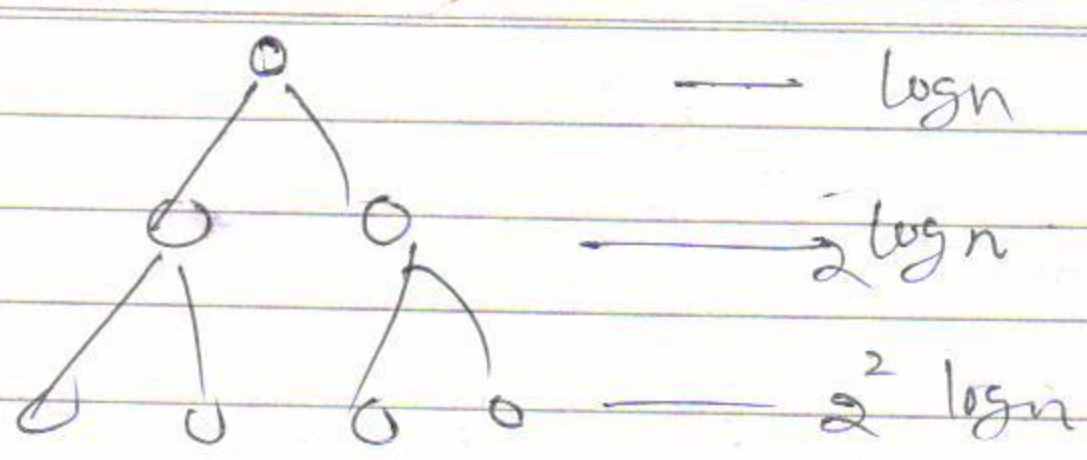
Wrost case

best case =>

$T(n-1) + T(n-2) + \log n$

266
Date

if-else
{
$T(n-1)$ a = f(n-1)
$T(n-2)$ b = f(n-2)
c = BS
}
$\log n$



$2^0 \log n + 2^1 \log n + 2^2 \log n + \cdots + 2^{n-1} \log n$

$= \log n \, (2^1 + 2^1 + 2^2)$ ⟶ GP series.

$= \log n \, (2^n)$

Q
```
S(n)
{
  for(i=0 to n-1)
  {
    a = a + s(i)
  }
}
```
⟹ n!

Now Basic over so

f(6).

Fib (6) ⇒ $2^6 - 1$ = function call

Fib (50) = $2^{50} - 1$ function call

Fib (10000 = $2^{10000} - 1$ function calls.

required

f(6)
↓
f(5)
↓
f(4)
f(3)
f(2)
f(1)

7 function calls required

Date

To evaluate Fib(6), acculy 7 function calls required but through above program, $2^6 - 1$ function call ase - again and again repeats.
In above recursive program. again and again many recursive calls are repeating which are overlapping subproblems.

* → Dynamic programming only call one function only once.

* → In the dynamic programming we will solve every function call only once. (means as soon as function value is evaluated it is stored in some table and it will be utilized again and again).

Q.
⇒ How many distinct function calls are there in Fibonacci of (n).

Ans.

Fib(6) = f(6), f(5), f(4), f(3), f(2), f(1), f(0) (6 distinct function calls.

Fib(n) = n, n-1, n-2, n-3, ... 2, 1, 0.

so ⟹ (n + 1) ⟹ 2 distinct function calls. are requised to evaluate Fibinucci of n.

⇒ Using Dynamic programming.

T(Fib(n)) = (n+1) * O(1)
= n+1

Time compleaity = O(n)

Date

with dp :-

Space Complexity = stack space + array

= O(n) + O(n)

= O(2n)

= O(n)

without dynamic programming :-

fib(n) = $O(2^n)$ → Time Complexity

fib(n) = O(n) { stack space } space Comp.

Q As soon as this program complete function call, store values in Table.

Now give T.C.

⇒ Means Dynamic Programming.

table

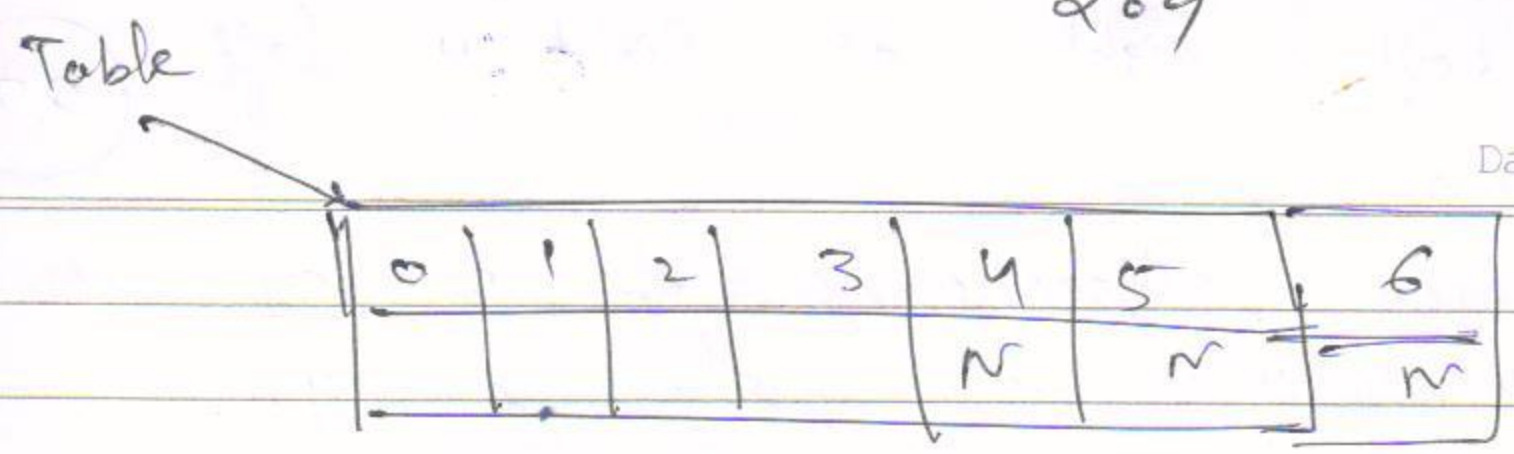
0	1	2	3	4	5	6
0	1	1	2	3	5	N

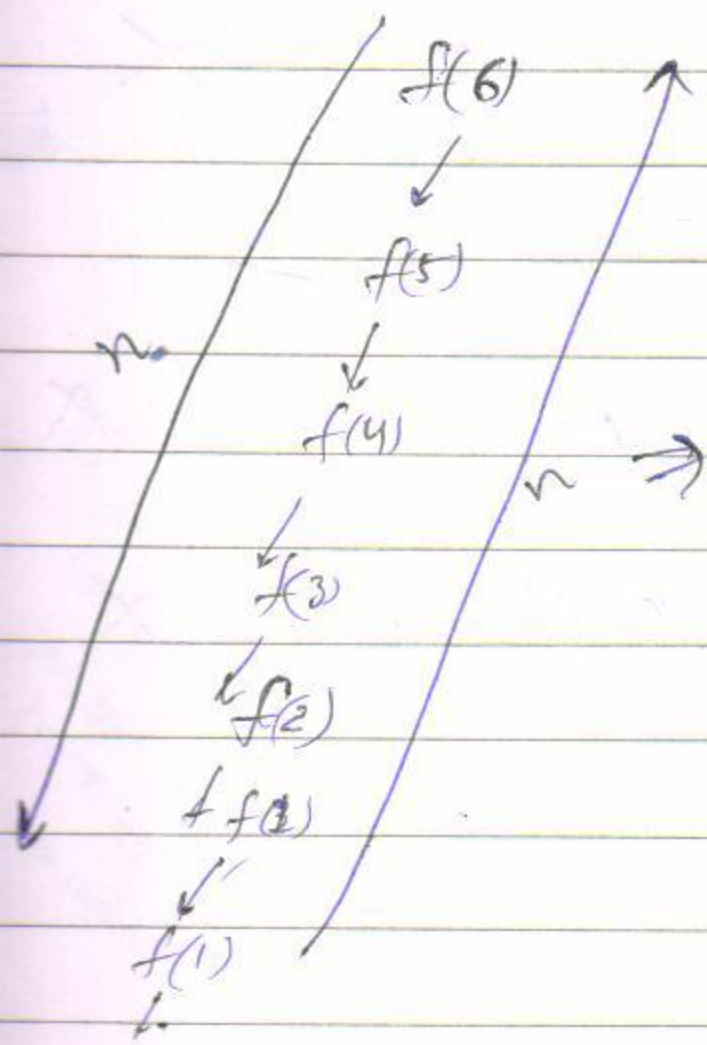
6 5 4

```
Fast_Fib(n)
{
    if(n==0 || n==1)  return(n);
    else
    {
        if(Table[n-1]==Null)
            Table[n-1] = fastfib(n-1)
        if(Table[n-2]==Null)
            Table[n-2] = fast-fib(n-2);
        Table[n] = Table[n^3-1] + Table[n^3-2];
        return(Table[n]);
    }
}
```

f(6)
f(5)
f(4)

Date





2N ⇒ $TC = O(n)$

Q. ⇒ Longest Common subsequance :-

⇒ subsequance :- Subsequance of of a given sequance is just the given sequance only in which 0 or more symbols are left out.

Ex:-

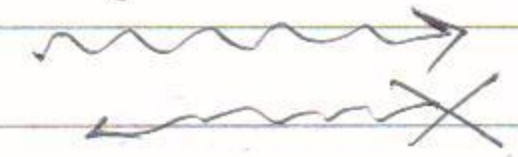
sequance $S = (A, B, B, A, B, B)$
$\quad\quad\quad

⇒ Backtracking is not allowed.
→ U may go Left to right or Right to left.

Date

⇒ Common Subsequance :-

Z is a common subsequance of 2 sequances x & y if z is subsequance to both x & y.
→ order can not be changed.



eg:- $x = (\underset{1}{A}, \underset{2}{B}, \underset{3}{B}, \underset{4}{A}, \underset{5}{B}, \underset{6}{B})$

$y = (\underset{1}{B}, \underset{2}{A}, \underset{3}{A}, \underset{4}{B}, \underset{6}{A}, \underset{6}{A})$

$CS_1 = ()$ ✓

$SS_2 = (A)$ ✓

$CS_3 = (A\ B)$ ✓

$CS_4 = (A\ B\ A)$ ✓

↳ Longest common subsequences.

$CS_5 = (A, B, AB)$ ✗

$CS_6 = (BA\ BA)$ ✗

Q.⇒ Find Longest common subsequance among given 2 sequances —

⇒ $x = (\underset{1}{A}, \underset{2}{B}, \underset{3}{A}\ \underset{4}{B}, \underset{6}{A}\ \underset{6}{A}, \underset{7}{B}\ \underset{8}{B}, \underset{9}{A})$

$y = (\underset{1}{B}, \underset{2}{A}, \underset{3}{B}, \underset{4}{B}, \underset{6}{A}\ \underset{6}{A}, \underset{7}{B}, \underset{8}{B}, \underset{9}{A})$

$CS_1 = ( )$

$CS_2 = (A\ B\ B\ A)$

$CS_{3,7} = (A\ B\ A\ A\ B\ B\ A)$

$CS_8 = (\underset{1}{A}\ \underset{2}{B}\ \underset{3}{B}\ \underset{4}{A}\ \underset{6}{A}\ \underset{7}{B}\ \underset{8}{B}\ A)$

↳ Longest Common subsequence.

$CS_1 = (A)$
$CS_2 = (A\ B)$
$CS_3 = (A\ B\ B)$
$CS_4 = (A\ B\ A\ B)$
$CS_5 = (A\ B\ A\ B\ A)$
$CS_6 = (B\ AA\ BB\ A)$

§ ⇒ How to find out Longest Common Subsequance — by Algo

Date

Ex 1: $x = (A_1, B_2, A_3, B_4, A)$

$y = (A_1\ B_2\ A_3\ B_4\ A)$

Ans $LCS(5,5) = 1 + LCS(4,4)$

$1 + LCS(3,3)$

$1 + LCS(2,2)$

$1 + LCS(1,1)$

$1 + LCS(0,0)$ → T.Condition

To store the sequance, take an array and insert that symbol into array whenever match.

Q. $x = (A_1\ A_2\ B_3\ A_4\ B_5\ A_6)$

$y = (A_1\ A_2\ A_3\ A_4\ B_5\ A_6)$

No Match

Ans $LCS(6,6) = 1 + LCS(5,5)$

$1 + LCS(4,4)$

$1 + LCS(3,3)$

$\max \begin{Bmatrix} LCS(3,2) \\ LCS(2,3) \end{Bmatrix}$

Recurrance Relation:-

Defination:-

$LCS(i,j)$ = the length of longest common sequence, possible with 2 sequances x, y where i is the no of element

Difference b/w OAC and OP:-

$OAC(n) = OAC(n/2)$

$OP(n) = OP(n-1)$

	OAC	OP
	100	100
	50	99
	25	97
	12	⋮
	⋮	

(both are recursive programms but)

Date

$$LCS(i,j) = \begin{cases} 0 & \text{if } i==0 \text{ (or) } j==0 \\ 1 + LCS(i-1, j-1) & \text{if } x[i]==y[j] \\ \max\{LCS(i, j-1),\ LCS(i-1, j)\} & \text{if } (x[i] \neq y[j]) \end{cases}$$

Recursive program :-

```
L.C.S. (i, j)
{
    if (i == 0 || j == 0) return (0);
    else
    {
        if (x[i] == y[j])
        return (1 + LCS (i-1, j-1));
        else
        {
            a = LCS (i, j-1);
            b = LCS (i-1, j);
            c = max (a, b);
            return (c);
        }
    }
}
```

T.C. $\Rightarrow O(2^{(i+j)}) = O(2^i \cdot 2^j)$

SC $\rightarrow$ stack space $= O(i+j)$

x = (A A A A)
    1 2 3 4

y = (B B B B B)
    1 2 3 4 5

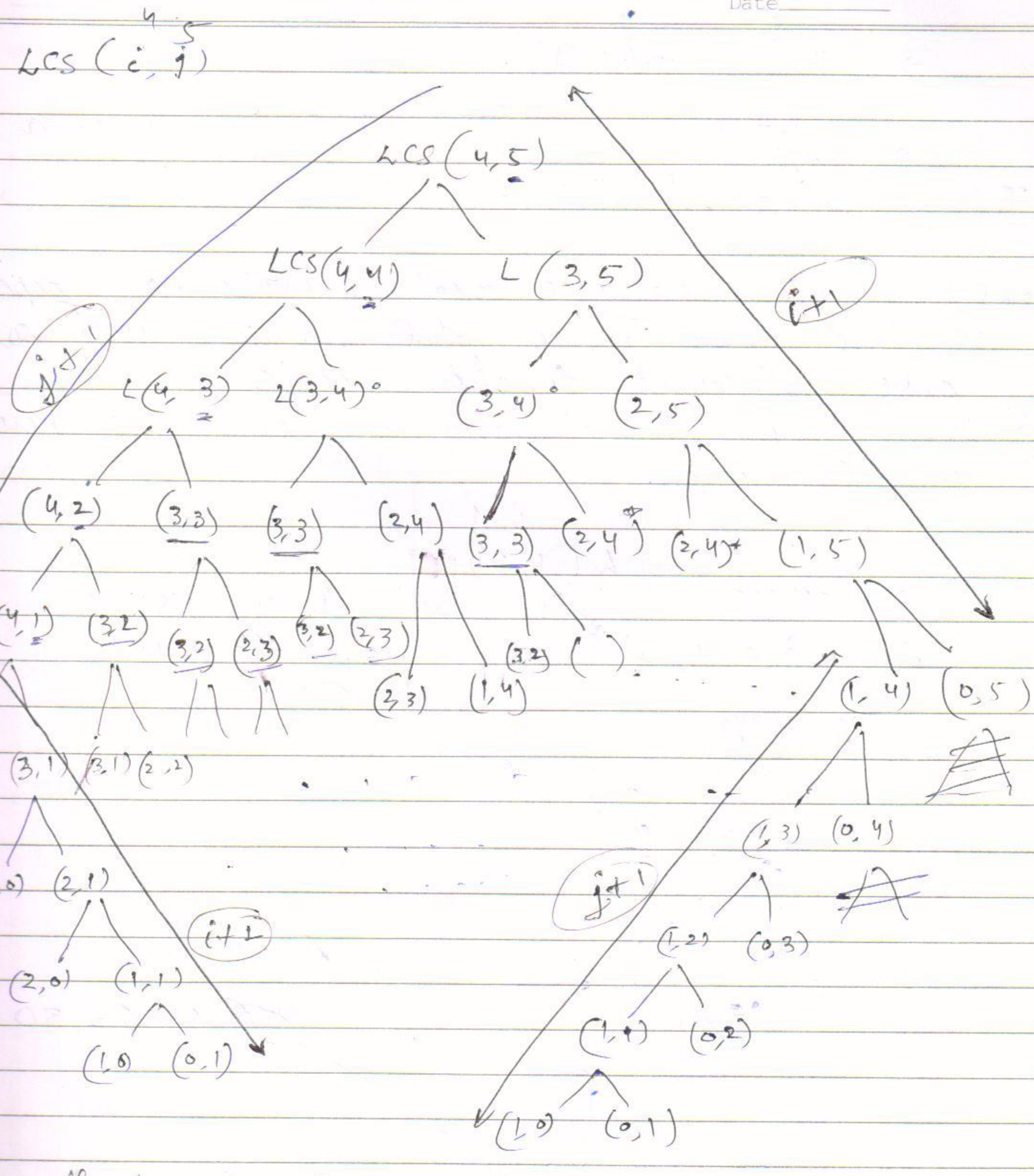
Without DP. :-

$LCS(m, n) \Rightarrow$

Time Complexity $= O(2^{m+n}) = 2^m \cdot 2^n$

$TC = O(2^n \cdot 2^n)$

Date



Length of tree = $i+j$.

LCS(i,j) will generate $(i+j)$ level complete binary tree.

so nodes = $(2^{i+j} - 1)$ nodes

= 2^{i+j} function calls

$\Rightarrow 2^{i+j} * [O(1)]$

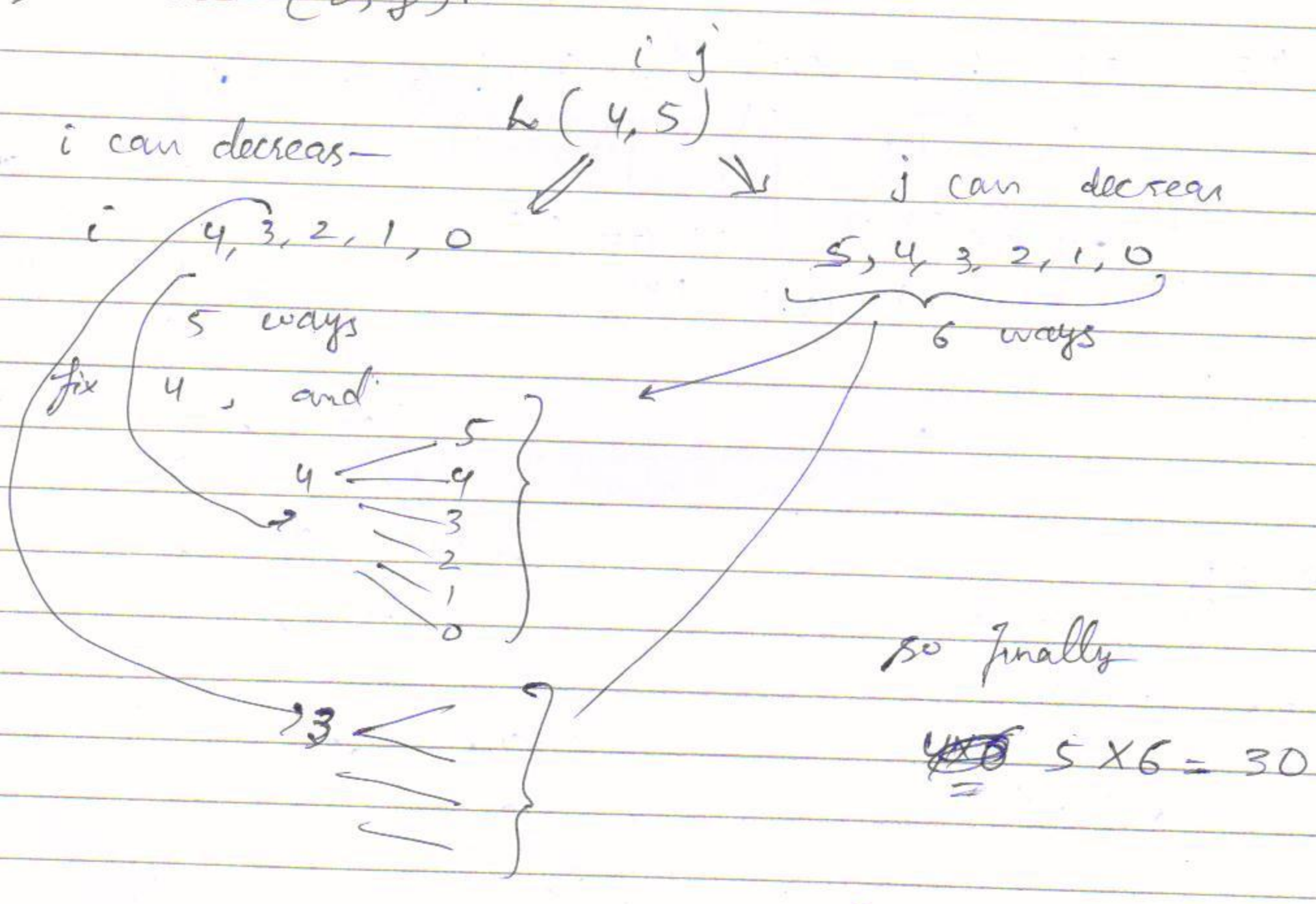
$\Rightarrow 2^{n+n}$ if $i=j=n$

$\Rightarrow 2^{n} \cdot 2^{n}$

$\underline{\text{T.C.}} = O(2^n \cdot 2^n)$

→ LCS of (i,j) recursive tree contains many repeatation so we have to go D.P., which will solve only distinct function calls.

Q. How many distinct function calls are there in LCS (i,j).



LCS $(m,n) = (m+1)\cdot(n+1)$ — Distinct function calls

$= mn$ ∀ fc

$= mn\,(O(1))$

$= O(m.n)$ if $m=n$

$= O(n^2)$

Date

$LCS(m,n)$ — m,n

without D.P

$i \quad j$

Time: $n+n$ — CBT

$n+n$

$= 2$

$= O(2^n \cdot 2^n)$

Space: $i+j$ level

$n+n$

$= 2n$

$= O(n)$

Stack Space

With D.P

Distinct function call

Time: $n \times n$

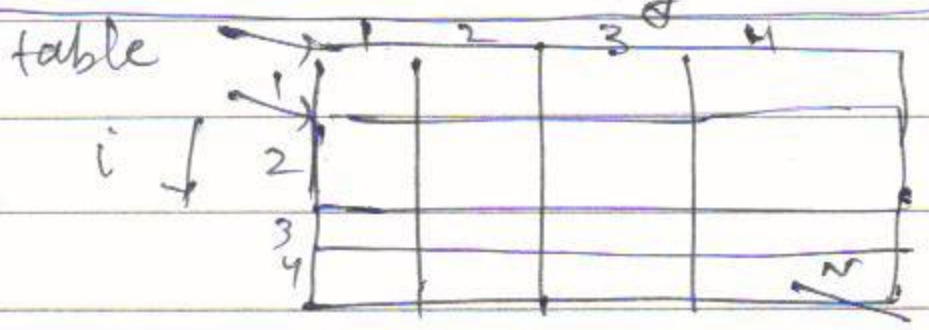
$= O(n^2)$

Space:-

$2n + n^2$

Stack space — Table

$SC = O(n^2)$




```
Fast LCS (i,j)
{
  if ( i==0 || j==0) return(0);
  else
    if ( x[i] == y[j] )
    {
      if (Table[i-1, j-1] == Null)
        Table[i-1, j-1] = Fast-LCS (i-1, j-1);
        Table[i,j] = 1 + Table[i-1, j-1];
        return ( Table [i,j]);
    }
  else
  { if ( Table [i, j-1] == Null)
      Table[i, j-1] = Fast-LCS(i, j-1);
    if ( Table [ i-1, j ] = Null)
```

Date

```
Table [i,j] = max { Table [i, j-1], Table [i-1, j-1] };
    return ( Table [i,j]);
  }
}
```